Master Thesis

Validierung und Erweiterung des Sicherheitskonzepts einer auf Web Services basierenden Infrastruktur

von

Dipl.-Inf. (FH) Peter Berthold
Matrikelnummer 150044

Hochschule Fulda
Fachbereich Angewandte Informatik
Marquardstrasse 35
36039 Fulda

Betreuer: Dipl.-Inform. Jan Peters

Prüfer: Prof. Dr. Hans-Theo Meinholz
Prof. Dr. Jan-Torsten Milde

Validierung und Erweiterung des Sicherheitskonzepts einer auf Web Services basierenden Infrastruktur

Aufgabenstellung zur Master Thesis von
Herrn Peter Berthold (Matr.-Nr. 150044)

In der Abteilung für Sicherheitstechnologie am Fraunhofer-Institut für Graphische Datenverarbeitung IGD wird an einer Java-basierten Sicherheitsplattform für Softwarekomponenten geforscht, die in ubiquitären Umgebungen Verwendung finden soll. Die Middleware, der im Rahmen des Projekts SicAri entwickelten und eingesetzten Plattform besteht aus drei Komponenten. Die Kommando-Shell wird vom Plattform-Administrator genutzt, um die Plattform zu starten sowie zur Laufzeit flexibel zu konfigurieren, zu erweitern und zu beobachten. Das so genannte Environment dient als Dienstumgebung für das lokale Service-Management. Damit bei der Interaktion von Nutzern, Anwendungen und Diensten die entsprechende rollenbasierte Sicherheitsrichtlinie durchgesetzt werden kann, wird nach der erfolgreichen Authentifikation eines Nutzers durch SmartCard, SoftToken oder Passwort diesem ein so genannter Sicherheitskontext zugeordnet.
Verschiedene Plattforminstanzen in der verteilten Infrastruktur sind in der Lage, auf der Ebene von Web Services miteinander zu kommunizieren, die während der Laufzeit automatisiert und transparent für den Programmierer und Anwender aus lokal geladenen Java-Komponenten generiert werden. Diese Web Services werden über einen externen Verzeichnisserver registriert und stehen dann innerhalb der verteilten Infrastruktur zur Verfügung. Das Framework zur Integration von Web Services basiert auf Apache AXIS und UDDI4J. Um Authentifikation und Autorisation bei der Nutzung entfernter Dienste zu realisieren, soll der lokale Sicherheitskontext unter Nutzung etablierter Standards der WS-Security-Familie auf die entfernte Plattform transferiert werden. Zentrale Komponenten der Infrastruktur und Vertrauensanker der zugrunde liegenden PKI bilden dabei u. a. der Identity Mananger bzw. der Authentication Manager.
Ziel der Master-Arbeit ist es, aufgrund einer initialen Bedrohungsanalyse die Sicherheitsinfrastruktur der Plattform im praktischen Einsatz zu validieren und durch Testmodule zu verifizieren, ob die in der Plattform-Spezifikation formulierten Sicherheitsanforderungen in der Implementierung korrekt umgesetzt wurden. Des Weiteren soll untersucht werden, inwiefern sich ggf. durch Erweiterung des Authentifikations- und Kommunikationskonzepts mobile Endgeräte anbinden lassen. Im Rahmen dieser Analyse sollen aktuelle Module durch neue Implementierungen ausgetauscht werden.
Die Master-Arbeit soll folgende Aspekte beinhalten:

- Literaturrecherche und Related Work
- Bedrohungsanalyse und Verifikation des Sicherheitskonzepts
- Analyse der neusten Web Service-Spezifikationen und Frameworks
- Software-Entwurf und Integration in die Plattform

Alle Anpassungen bzw. offene Probleme des Sicherheitskonzepts der Plattform sollen mit Referenz auf die ursprüngliche Spezifikation explizit heraus gestellt werden.

Jan Peters — Darmstadt, 15.10.2006

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Master Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Fulda, Juli 2007 Peter Berthold

Danksagung

Hiermit möchte ich mich bei allen Mitwirkenden, die das Entstehen dieser Arbeit gefördert haben, bedanken. Dazu gehören zunächst mein Betreuer Jan Peters am Fraunhofer Institut sowie mein Referent Prof. Dr. Meinholz an der Hochschule Fulda. Beiden möchte ich für ihre konstruktive Kritik und die gute Zusammenarbeit danken.

Meiner Frau Karina möchte ich für die Geduld beim ersten Korrekturlesen und die aufmunterten Worte danken, wenn es mal nicht so voranging wie es geplant war. Außerdem auch Frau Katja Mattern für ihre Mühe beim zweiten Korrekturlesen.

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Motivation . . . 1
- 1.2 Typografie . . . 1
- 1.3 Aufbau . . . 2

2 Grundlagen **3**
- 2.1 IT-Sicherheit . . . 3
  - 2.1.1 IT-Grundbegriffe . . . 3
  - 2.1.2 Schutzziele . . . 5
    - 2.1.2.1 Authentizität . . . 6
    - 2.1.2.2 Integrität . . . 7
    - 2.1.2.3 Vertraulichkeit . . . 7
    - 2.1.2.4 Verfügbarkeit . . . 7
    - 2.1.2.5 Verbindlichkeit . . . 8
- 2.2 Bewertungskriterien . . . 8
  - 2.2.1 TCSEC-Kriterien . . . 9
  - 2.2.2 IT-Kriterien . . . 10
    - 2.2.2.1 Funktionsklassen . . . 10
    - 2.2.2.2 Qualitätsstufen . . . 11
  - 2.2.3 ITSEC . . . 13
  - 2.2.4 Common Criteria . . . 17
    - 2.2.4.1 Einführung und allgemeines Modell . . . 19
    - 2.2.4.2 Schutzprofile . . . 19
    - 2.2.4.3 Sicherheitsvorgaben . . . 20
    - 2.2.4.4 Funktionale Sicherheitsanforderungen . . . 21
    - 2.2.4.5 Anforderungen an die Vertrauenswürdigkeit . . . 23
- 2.3 Serviceorientierte Architekturen . . . 25
  - 2.3.1 Interaktion in einer serviceorientierten Architektur . . . 26
  - 2.3.2 Merkmale von serviceorientierten Architekturen . . . 27
- 2.4 Web Services . . . 29
  - 2.4.1 XML . . . 30
  - 2.4.2 XML-Programmiermodelle . . . 32
    - 2.4.2.1 Simple API for XML . . . 32
    - 2.4.2.2 Document Object Model . . . 34
    - 2.4.2.3 Streaming API for XML . . . 34
  - 2.4.3 SOAP . . . 35
    - 2.4.3.1 SOAP-Verarbeitungsmodell . . . 36

2.4.3.2 SOAP-Faults . . . 37
2.4.4 WSDL . . . 37
2.4.5 UDDI . . . 39
2.5 Single Sign-On . . . 39
2.5.1 Kerberos . . . 40
2.5.2 Microsoft Passport . . . 43
2.5.3 Liberty Alliance Project . . . 46
2.5.4 Security Assertions Markup Language . . . 46
2.5.4.1 SSO mit SAML . . . 48
2.6 Kryptografische Grundlagen . . . 49
2.6.1 Verschlüsselung und Entschlüsselung . . . 50
2.6.2 Hash-Funktionen . . . 55
2.6.3 Elektronische Signaturen . . . 56
2.6.4 Digitale Zertifikate . . . 57
2.7 XML Sicherheit . . . 58
2.7.1 XML-Signature . . . 58
2.7.2 XML-Encryption . . . 61
2.7.3 XML-Firewalls . . . 66
2.8 Web Service-Sicherheit . . . 67
2.8.1 Sicherheitsmechanismen auf Transportebene . . . 67
2.8.2 WS* . . . 68
2.8.3 WS-Security . . . 69
2.8.3.1 WSS-Security Token . . . 71
2.8.3.2 Digitale Signaturen . . . 74
2.8.3.3 Verschlüsselung . . . 75
2.8.3.4 Fehlerbehandlung innerhalb von WS-Security . . . 77
2.8.4 WS-Policy . . . 77
2.8.5 WS-Trust . . . 78
2.8.6 WS-Privacy . . . 80
2.8.7 WS-SecureConversation . . . 81
2.8.8 WS-Federation . . . 83
2.8.9 WS-Authorization . . . 84
2.8.10 WS-PolicyAttachement . . . 84
2.8.11 WS-SecurityPolicy . . . 85
2.8.12 Extensible Access Control Markup Language . . . 86
2.8.13 XML Key Management Specification . . . 86
2.8.14 Weitere WS*-Spezifikationen . . . 87
2.9 Java ME . . . 89
2.9.1 Übersicht über die Java ME . . . 89
2.9.1.1 CLDC . . . 89
2.9.1.2 CDC . . . 90
2.9.1.3 Optionale APIs . . . 91
2.9.2 Generic Connection Framework . . . 92
2.9.3 Vergleich der Java ME mit der Java SE . . . 92

3 SicAri **95**
3.1 Überblick zur SicAri-Plattform . . . 95
3.2 Anwendungsbereiche der Plattform . . . 96
3.3 Sicherheitsanforderungen . . . 97
3.4 Architektur der Plattform . . . 100
3.4.1 SicAri-Kernel . . . 102
3.4.1.1 Shell . . . 102
3.4.1.2 Environment . . . 102
3.4.1.3 Sicherheitskontext . . . 103
3.4.2 Sicherheitsarchitektur der SicAri-Plattform . . . 105
3.4.2.1 Komponenten des SicAri-Sicherheits-Frameworks . . . 105
3.4.2.2 RBAC als Sicherheitsmodell . . . 105
3.4.2.3 Referenzmonitor-Modell . . . 106
3.4.3 Plattform-Kommunikation . . . 107
3.4.3.1 SicAri-WS-Framework und Apache Axis . . . 107
3.5 Basisdienste der SicAri-Plattform . . . 110
3.5.1 Security Policy Architektur . . . 110
3.5.2 Kontextmanager . . . 111
3.5.3 Authentifikationsmanagement . . . 111
3.5.4 Identity Management . . . 113
3.5.4.1 Identity Administration . . . 113
3.5.4.2 Identity Manager . . . 114
3.5.5 Key Manager . . . 115
3.5.6 Persistency Service . . . 115
3.6 Anwendungsdienste der SicAri-Plattform . . . 116
3.6.1 Sensormodule . . . 116
3.6.2 Kryptografische Funktionen . . . 116
3.6.3 Kommunikationsprotokolle . . . 116
3.6.4 Web Services . . . 117

4 Realisierung **119**
4.1 Konzeption . . . 119
4.1.1 Authentifikation . . . 119
4.1.1.1 JAAS . . . 119
4.1.1.2 Security Token und Soft Token . . . 121
4.1.1.3 Authentifikationsmethoden & -ebenen . . . 122
4.1.1.4 Authentifikationsszenarien . . . 123
4.1.1.5 Protokollspezifikation $AP1 : (S1_{global}, M1\text{-}M3, L1/L2)$ . . . 126
4.1.1.6 Plattform-Authentifikation . . . 129
4.1.2 Sicherheitsenforcement . . . 129
4.1.2.1 Protokollspezifikation $AP2 : (S1_{global}, M4, L6)$ . . . 130
4.1.3 Mobile Endgeräte . . . 131
4.1.3.1 Protokollspezifikation $AP3 : (S2_{remote}, M4, L6)$ . . . 132
4.2 Implementierung . . . 134
4.2.1 Authentifikation . . . 136
4.2.1.1 Lokale Authentifikation . . . 138
4.2.1.2 Globale Authentifikation . . . 139
4.2.1.3 SecurityToken . . . 141

4.2.1.4 Authentifikationsmanager . . . 145
4.2.1.5 AuthenticationManagerHandler . . . 146
4.2.2 Sicherheitsenforcement . . . 151
4.2.2.1 SecurityTokenHandler . . . 151
4.2.3 ST Übertragung . . . 155
4.2.4 SOAP-Body Verschlüsselung und Signatur . . . 156
4.2.5 Validierung der SSL/TLS-Verbindung . . . 157
4.2.6 Identifikation der öffentlichen Chiffrierungsschlüssel . . . 157
4.2.7 XPath gesteuerte Verarbeitung . . . 159
4.2.8 Fehlerbehandlung . . . 162
4.2.9 Mobile Endgeräte . . . 163
4.2.9.1 Optionale APIs . . . 163
4.2.9.2 WS-Frameworks . . . 164
4.2.9.3 Entwicklungswerkzeuge & Konfiguration . . . 165
4.2.9.4 Evaluation der Authentifikationsszenarien . . . 167
4.3 Integration . . . 172
4.3.1 Service-Bootstrapping . . . 172
4.3.1.1 whatis.conf . . . 173
4.3.1.2 rc.conf . . . 173
4.3.1.3 sicari.conf . . . 174
4.3.1.4 rc.dependencies . . . 174
4.3.1.5 rc.admin . . . 174
4.3.1.6 rc.network . . . 175
4.3.1.7 webservice.conf . . . 176
4.3.1.8 login.conf . . . 176
4.3.2 WS-Deployment . . . 177
4.3.3 WS-Suche . . . 178
4.3.4 Handler-Deployment . . . 179

5 Bedrohungsanalyse 181
5.1 Methodik . . . 181
5.1.1 Grundbegriffe: Schwachstelle, Bedrohung und Angriff . . . 181
5.1.2 Bedrohungsbaum . . . 183
5.1.3 Kategorisierung von Bedrohungen . . . 184
5.1.4 Bedrohungen . . . 186
5.2 Durchführung . . . 187
5.2.1 Identifikation der möglichen Bedrohungen . . . 187
5.2.1.1 Externe Komponenten . . . 188
5.2.1.2 Interne Komponenten einer Plattform . . . 189
5.2.1.3 Interne Services einer Plattform . . . 190
5.2.1.4 Authentifikationsmethoden einer Plattform . . . 194
5.2.1.5 Weitere Angriffspunkte . . . 198
5.2.2 Abwehr der Bedrohungen . . . 199
5.2.2.1 Externe Komponenten . . . 199
5.2.2.2 Interne Komponenten und Services einer Plattform . . . 200
5.2.2.3 Authentifikationsmethoden einer Plattform . . . 203
5.2.2.4 Weitere Angriffspunkte . . . 207
5.3 Risikoanalyse . . . 208

6 Evaluation **211**
6.1 Leistungsfähigkeitstests . . . 211
6.1.1 Authentifikation . . . 212
6.1.1.1 Testszenario . . . 212
6.1.1.2 Messpunkte . . . 213
6.1.1.3 Ergebnisse . . . 215
6.1.1.4 Fazit . . . 219
6.1.2 SSO . . . 219
6.1.2.1 Testszenario . . . 219
6.1.2.2 Messpunkte . . . 220
6.1.2.3 Ergebnisse . . . 221
6.1.2.4 Fazit . . . 225

7 Abschlussbetrachtung **227**
7.1 Fazit . . . 227
7.2 Ausblick . . . 228

A Akronyme **231**

B WS-Frameworks **237**

C Security Token-Struktur **239**
C.1 ASN1-Security Token . . . 239
C.1.1 ASN1SecurityToken . . . 239
C.1.1.1 PlainSecurityToken . . . 239
C.1.1.2 SignedData . . . 239
C.1.1.3 SignerInfo . . . 240
C.1.1.4 AlgorithmIdentifier . . . 240
C.1.2 ASN1EncryptedSecurityToken . . . 240
C.1.2.1 EnvelopedData . . . 241
C.1.2.2 RecipientInfo . . . 241
C.1.2.3 EncryptedContentInfo . . . 241
C.1.2.4 AlgorithmIdentifier . . . 241
C.2 SAMLSecurityToken . . . 242
C.2.1 SAML-Assertion . . . 242
C.2.2 SAML-Assertion-Beispiel . . . 244

D Nachrichtenmitschnitte **245**
D.1 SOAP-Nachrichten mit SecurityToken . . . 245
D.1.1 SOAP-Nachricht mit ASN1SecurityToken . . . 245
D.1.2 SOAP-Nachricht mit ASN1EncryptedSecurityToken . . . 245
D.1.3 SOAP-Nachricht mit SAMLSecurityToken . . . 246
D.1.4 SOAP-Nachricht mit chiffriertem SAMLSecurityToken . . . 248
D.2 Aufruf eines Web Service mit ST und WSS-Mechanismen . . . 249
D.2.1 Hello-WS mit ASN.1-ST . . . 249
D.2.2 Hello-WS mit ASN.1-ST & SOAP-Body Verschlüsselung . . . 250
D.2.3 Hello-WS mit ASN.1-ST & SOAP-Body Signatur . . . 251
D.2.4 Hello-WS mit ASN.1-ST & SOAP-Body Verschlüsselung und Signatur 252
D.3 SOAP-Nachrichten der WS-Methoden des Authentifikationsmanagers . . . 255

D.3.1 getAuthenticationMethods(String userID) . . . 255
D.3.1.1 Anfrage . . . 255
D.3.1.2 Antwort . . . 255
D.3.2 authenticate(byte m, String userID, byte[] req) . . . 255
D.3.2.1 Anfrage . . . 255
D.3.2.2 Antwort . . . 256
D.3.3 cancel(String userID) . . . 257
D.3.3.1 Anfrage . . . 257
D.3.3.2 Antwort . . . 257
D.3.4 authenticate(String tokenId) . . . 257
D.3.4.1 Anfrage . . . 257
D.3.4.2 Antwort . . . 258
D.3.5 authenticate(String tokenId, String tokenType) . . . 258
D.3.5.1 Anfrage . . . 258
D.3.5.2 Antwort . . . 259
D.3.6 SOAP-Fault . . . 261

E Konfigurationsdateien und -skripte 263
E.1 Neue Konfigurationsdateien . . . 263
E.1.1 authentication.conf . . . 263
E.1.2 ws-st-handler.conf . . . 263
E.1.3 ws-am-handler.conf . . . 264
E.2 Erweiterte Konfigurationsdateien und -skripte . . . 266
E.2.1 login.conf . . . 266
E.2.2 rc.admin . . . 267
E.2.3 rc.conf . . . 268
E.2.4 rc.dependencies . . . 270
E.2.5 rc.main . . . 274
E.2.6 rc.network . . . 276
E.2.7 sicari.conf . . . 279
E.2.8 webservice.conf . . . 279
E.2.9 whatis.conf . . . 280

F Inhalt der CD 283
F.1 Übersicht . . . 283
F.2 Hierarchie . . . 283

Abbildungsverzeichnis

2.1 Schichtenmodell der IT-Sicherheit 6
2.2 Abhängigkeiten zwischen IT-Kriterienkatalogen 9
2.3 Funktionsklassen der Kriterienkataloge TCSEC, ITSK und ITSEC 11
2.4 Gegenüberstellung der Evaluationsstufen zum Aufbau der Kriterien 15
2.5 Korrektheitskriterien der Evaluationsstufen eines Entwicklungsprozesses 16
2.6 Struktureller Aufbau eines Schutzprofils 21
2.7 Aufbau der Funktionsklassen der Common Criteria 22
2.8 Die Rollen in einer serviceorientierten Architektur 27
2.9 Web Services-Architektur 30
2.10 Web Services-Schichtenmodell 31
2.11 Aufbau von XML-Dokumenten 32
2.12 SAX-Übersicht 33
2.13 Einlesen eine XML-Dokument mittels SAX-Parser 33
2.14 DOM-Übersicht 34
2.15 Einlesen eine XML-Dokument mittels DOM-Parser 34
2.16 Einlesen eines XML-Dokuments mittels StAX-Cursor-Verarbeitung 35
2.17 SOAP innerhalb des TCP/IP Protokollstapels 36
2.18 SOAP Nachrichtenaufbau 36
2.19 Das SOAP-Fault Element 38
2.20 Aufbau eines WSDL-Dokuments 38
2.21 Kerberos Architektur und Protokollschritte 41
2.22 Single Sign-On mit Passport 44
2.23 Bestandteile von SAML 47
2.24 SAML-SSO: Web Browser SSO Profile 49
2.25 Symmetrische Verschlüsselung 52
2.26 Asymmetrische Verschlüsselung 53
2.27 Genereller Aufbau von XML-Signature 59
2.28 Beispiel einer XML-Signatur 60
2.29 XML-Signature mit Angabe eines RSA-Schlüssels 61
2.30 Enveloped- und Enveloping XML-Signature 62
2.31 Genereller Aufbau von XML-Encryption 63
2.32 Verschlüsselung mit Angaben des Schlüssels und der Verschlüsselungsmethode 63
2.33 Hybride Verschlüsselung 64
2.34 Ein XML-Dokument als Ausgangsbeispiel für eine XML-Verschlüsselung . . . 64
2.35 XML-Encryption des kompletten Dokuments 65
2.36 XML-Encryption auf Element-Ebene 65
2.37 XML-Encryption auf Content-Ebene 66

2.38 Web Services-Sicherheit Roadmap . . . 69
2.39 WSS-UsernameToken eingebettet in einer SOAP-Nachricht . . . 72
2.40 WSS X.509-BinarySecurityToken . . . 72
2.41 Beispiel einer SecurityTokenReference . . . 73
2.42 WSS-Signatur und STR Dereference Transformation . . . 75
2.43 WSS-Verschlüsselung . . . 76
2.44 Beispielstruktur einer WS-Policy . . . 78
2.45 WS-Trust-Modell . . . 79
2.46 Allgemeiner Aufbau des RequestSecurityToken . . . 79
2.47 Allgemeiner Aufbau des RequestSecurityTokenResponse . . . 80
2.48 Aufbau einer WS-SecureConversation . . . 82
2.49 Allgemeiner Aufbau des SecurityContextToken . . . 82
2.50 Topologie eines Federation Modells . . . 83
2.51 WS-SecurityPolicy - Protection Assertions . . . 85
2.52 WS-ReliableMessaging Modell . . . 88

3.1 High-Level Architektur der SicAri-Plattform . . . 96
3.2 Architektur der SicAri-Plattform . . . 101
3.3 Zugriffskontrolle und Sessionkontext . . . 104
3.4 Komponenten des SicAri-Sicherheits-Frameworks . . . 105
3.5 SicAri Referenz-Monitor . . . 106
3.6 Axis im Web Container . . . 108
3.7 Axis Pivot-Handler . . . 108
3.8 Axis-Nachrichtenpfad . . . 109
3.9 Komponenten der SicAri Security-Policy Architektur . . . 110
3.10 Komponenten des SicAri-Authentifikationsmanagement . . . 112
3.11 Komponenten des SicAri-Identitätsmanagement . . . 114

4.1 Beteiligte Komponenten der Benutzerauthentifikation . . . 120
4.2 Sequenzdiagramm: Benutzerauthentifikation mit dem JAAS-Framework . . . 121
4.3 Lokale Authentifikationsszenarien $S1_{local} - S3_{local}$ . . . 124
4.4 Globale Authentifikationsszenarien $S1_{global} - S3_{global}$ . . . 125
4.5 Entfernte Authentifikationsszenarien $S1_{remote} - S2_{remote}$ . . . 125
4.6 Beteiligte Komponenten des Sicherheitsenforcement . . . 130
4.7 Klassendiagramm: Paket de.sicari.authentication . . . 135
4.8 Klassendiagramm: Zusammenspiel von JAAS und SicAri . . . 136
4.9 Klassendiagramm: SicAri-Login-Module . . . 138
4.10 Sequenzdiagramm: Globale Authentifikation mittels Soft Token . . . 140
4.11 Klassendiagramm: SecurityToken-Übersicht . . . 141
4.12 Klassendiagramm: SecurityToken-detailliert . . . 144
4.13 Klassendiagramm: SicAri-Authentication-Services . . . 146
4.14 Der AuthenticationManagerHandler im Authentifikationsprozess . . . 148
4.15 Klassendiagramm: Axis-Handler . . . 152
4.16 Der `SecurityTokenHandler` im SSO-Prozess . . . 153
4.17 Klassendiagramm: SOAPSecurity . . . 157
4.18 Klassendiagramm: SSLSessionVerifier . . . 158
4.19 Klassendiagramm: XPathEval . . . 160
4.20 Anfragenachricht zur Authentifikation . . . 162

4.21 Antwortnachricht zur Authentifikation 162
4.22 Web Service-Zugriff unter Verwendung des WSOAP-Framework 166
4.23 Berechnung der Soft Token-basierenden Authentifikationsanfrage unter der CDC 168
4.24 Authentifikation unter Verwendung des WSOAP-Framework 169
4.25 SSO unter Verwendung des WSOAP-Framework 170
4.26 Service-/WS-Deployment 177
4.27 Service-/WS-Discovery 178

5.1 Klassifikation von Gefährdungsfaktoren 182
5.2 IT-Sicherheitsprozess 183
5.3 Grundlegender Aufbau eines Bedrohungsbaums 184
5.4 Bedrohungsbaum: UDDI-Registry 189
5.5 Bedrohungsbaum: SicAri-Services 191
5.6 Bedrohungsbaum: Authentifikationsmanager 193
5.7 Bedrohungsbaum: Benutzername/Password-Authentifikation ($M1$) 194
5.8 Bedrohungsbaum: Soft Token-Authentifikation ($M2$) 195
5.9 Bedrohungsbaum: Smart Card-Authentifikation ($M3$) 196
5.10 Bedrohungsbaum: Security Token-Authentifikation ($M4$) 197

6.1 Messpunkte im Nachrichtenfluss der Authentifikation 214
6.2 Durchschnittliche Gesamtlaufzeit: Authentifikation 216
6.3 Durchschnittliche Bearbeitungszeit: AMH 217
6.4 Durchschnittliche Übertragungszeit: Authentifikation 218
6.5 Messpunkte im Nachrichtenfluss des SSO 221
6.6 Durchschnittliche Gesamtlaufzeit: SSO 222
6.7 Durchschnittliche Bearbeitungszeit: STH 224
6.8 Durchschnittliche Übertragungszeit: SSO 225

Tabellenverzeichnis

2.1 Stufen der ITSK zur Bewertung der Stärke der Mechanismen 12
2.2 Qualitätsstufen der ITSK 13
2.3 Vertrauenswürdigkeitskriterien bei CC und ITSEC 18
2.4 Anwendung der drei Teile der CC für die jeweiligen Zielgruppen 19
2.5 Vertrauenswürdigkeitsklassen und -familien 24
2.6 Vertrauenswürdigkeitsfamilien und Evaluierungsstufen 25
2.7 Symmetrische Verschlüsselungsalgorithmen 53
2.8 Web Service-Namensräume 71
2.9 WS-Security SOAP-Faults 77
2.10 Java ME - Aufbau der Connected Limited Device Configuration 90
2.11 Java ME - Aufbau der Connected Device Configuration 91
2.12 Java ME - Optionale Pakete 91
2.13 Java ME - Vergleich CLDC und Java SE 1.3.1 93
2.14 Java ME - Vergleich CDC und Java SE 1.4.2 94

3.1 Funktionale Sicherheitsanforderungen der SicAri-Plattform 100

4.1 Syntax der Protokollspezifikation $AP1$ 126
4.2 Syntax der Protokollspezifikation $AP2$ 131
4.3 Mobile Authentifikation 132
4.4 Syntax der Protokollspezifikation $AP3$ 132
4.5 Authentifikationsszenarien CLDC 172
4.6 Authentifikationsszenarien CDC 172

5.1 Zuordnung Bedrohungen und Abwehrmaßnahmen: Externe Komponenten . . . 199
5.2 Zuordnung Bedrohungen und Abwehrmaßnahmen: Interne Komp. & Services . 201
5.3 Zuordnung Bedrohungen und Abwehrmaßnahmen: Authentifikationsmethoden 204
5.4 Zuordnung Bedrohungen und Abwehrmaßnahmen: Weitere Bedrohungen . . . 207

6.1 Ergebnisse Soft Token-Authentifikation: Gesamtlaufzeit 216
6.2 Ergebnisse Soft Token-Authentifikation: Bearbeitungszeit Axis-Handler AMH . 217
6.3 Ergebnisse Soft Token-Authentifikation: Übertragungszeit 218
6.4 Antwortnachrichtengröße und Overhead durch SOAP 220
6.5 Bearbeitungszeit SSO 222
6.6 Ergebnisse SSO: Gesamtlaufzeit 223
6.7 Ergebnisse SSO: Bearbeitungszeit Axis-Handler STH 224
6.8 Ergebnisse SSO: Übertragungszeit 225

B.1 Generelle Features der existierenden WS-Frameworks 237
B.2 Unterstützte JSR der existierenden WS-Frameworks 238
B.3 Unterstützte WS*-Standards der existierenden WS-Frameworks 238

Kapitel 1

Einführung

1.1 Motivation

Durch das Aufkommen der *Web Services (WS)*-Technologie haben serviceorientierte Architekturen (*Service Oriented Architecture (SOA)*) neuen Auftrieb erhalten. Unternehmen können damit auf einfache Weise verschiedene Dienste für Nutzer über das Internet bereitstellen und damit ihre Geschäftsprozesse automatisieren und vereinfachen.
Durch die Bereitstellung von Funktionalitäten als Services haben Nutzer jedoch auch einen direkten Zugriff und Einfluss auf die wertschöpfenden Prozesse eines Unternehmens. Dabei ist es zwingend, die Sicherheitsanforderungen eines Unternehmens zu berücksichtigen, um Spionageangriffen und anderen geschäftsschädigenden Aktivitäten entgegenwirken zu können.
Um Sicherheitsaspekte, wie beispielsweise Authentizität, Integrität oder Vertraulichkeit, zu gewährleisten, sind im WS-Umfeld viele verschiedene Sicherheitsstandards bekannt. Wie diese eingesetzt werden können und welche davon sinnvoll sind, soll im Rahmen dieser Arbeit im praktischen Einsatz untersucht werden. Dabei soll auch überprüft werden, inwiefern mobile Endgeräte auf Web Services zugreifen, und wie dabei die gestellten Sicherheitsanforderungen gewährleistet werden können.
Dies soll dabei anhand der Middleware-Plattform *SicAri*[1] untersucht werden. Die SicAri-Plattform stellt eine auf Java-Technologie aufbauende Architektur für die sichere ubiquitäre Nutzung von Internetanwendungen bereit.

1.2 Typografie

Klassennamen und Datentypen sowie Quelltextauszüge sind in dieser Arbeit in `Schreibmaschinenschrift` gesetzt. Neu eingeführte Begriffe sind im Text an ihrer *kursiven* Darstellung zu erkennen. Abkürzungen werden bei ihrem erstmaligen Auftreten explizit erklärt und anschließend ohne weitere Hervorhebung verwendet.
Englische Begriffe werden in dieser Arbeit ohne besondere Hervorhebung verwendet. Existieren adäquate deutsche Übersetzungen, so werden diese stattdessen verwendet, wobei beim ersten Auftreten der korrespondierende englische Begriff in Klammern und kursiver Schreibweise genannt wird. Ggf. wird bei Begriffen die im normalen Sprachgebrauch nicht eindeutig sind, eine kurze Beschreibung als Fußnote eingefügt, die dessen Bedeutung im Kontext der Arbeit definiert.

[1] SicAri: Eine **Sic**herheits**ar**chitektur und deren Anwendungen für die ubiquitäre **I**nternetnutzung, `http://www.sicari.de`

Zusammengesetzte Begriffe aus mehreren englischen Wörtern werden in dieser Arbeit ohne Bindestrich verwendet. Angesetzte deutsche Nomen werden mit Bindestrich angefügt.

1.3 Aufbau

Die vorliegende Arbeit besteht inklusive diesem Kapitel aus insgesamt sieben Kapiteln. Zu Beginn jedes Kapitels wird ein kurzer Überblick über dessen Inhalt gegeben.
Im zweiten Kapitel werden die für das weitere Verständnis benötigten Grundlagen, wie beispielsweise IT-Sicherheitsgrundbegriffe, das Konzept der serviceorientierten Architektur, kryptografische Grundlagen und Web Service Techniken, vorgestellt. Außerdem werden Kriterienkataloge zur Bewertung der IT-Sicherheit und Techniken zur Wahrung der Sicherheit von XML- und Web Service-Nachrichten erklärt.
Im dritten Kapitel erfolgt ein Überblick über die SicAri-Plattform. Hier werden das Konzept und die Architektur der Plattform vorgestellt.
Das Kapitel 4 umfasst die im Rahmen dieser Arbeit erstellten Implementierungen. Hier wird die Umsetzung der erweiterten Authentifikationssmechanismen durch das auf Web Services basierende Protokoll vorgestellt und die Mechanismen für die Implementierung des SSO beschrieben. Außerdem wird gezeigt, wie mobile Endgeräte auf Basis der Java ME angebunden werden können. Das Kapitel zeigt zudem, wie die implementierten Komponenten in die bestehende SicAri-Architektur integriert werden.
Im Kapitel 5 werden die bestehenden und erweiterten Authentifikationssmechanismen der Plattform hinsichtlich möglicher Bedrohungen untersucht. Im Kapitel „Evaluation“ werden schließlich die implementierten Erweiterungen auf ihre Leistungsfähigkeit untersucht.
Die Arbeit schließt mit einem Ausblick und Fazit.
Im Anhang werden die verwendeten Akronyme aufgeführt, ein Vergleich der verschiedenen WS-Frameworks vorgenommen sowie ein detaillierter Überblick über die Struktur der Security Tokens gegeben. Außerdem werden eine Reihe von SOAP-Nachrichten aufgeführt, die bei dem implementierten auf Web Services basierenden Protokoll ausgetauscht werden. Schließlich werden noch die erstellten und angepassten Konfigurationsdateien und -skripte der SicAri-Plattform gelistet und der Inhalt der beigelegten CD erläutert.

Kapitel 2

Grundlagen

In diesem Kapitel werden die für die weitere Arbeit benötigten Grundlagen erläutert. Zunächst werden Grundbegriffe der IT-Sicherheit vorgestellt und Schutzziele definiert. Anschließend werden Kriterienkataloge für die Bewertung der Sicherheitsfunktionen eines IT-Systems behandelt. Nach den Merkmalen einer Service-orientierten Architektur werden schließlich die Grundlagen von Web Services vorgestellt. Hierbei wird auf die Technologien XML, SOAP, WSDL und UDDI eingegangen. Anschließend wird das Konzept des SSO an den Beispielen Kerberos und Passport erklärt und gezeigt, wie sich mit der XML-Technologie SAML der SSO umsetzen lässt. Schließlich werden kryptografische Grundlagen wie Verschlüsselung, Signatur, Hash-Funktion und digitale Zertifikate vorgestellt und aufgezeigt, wie sich diese Techniken auf XML-Dokumente anwenden lassen. Anschließend wird noch detailliert auf die Sicherheit von Web Services und die damit verbunden WS*-Spezifikationen eingegangen. Das Kapitel schließt mit einem kurzen Überblick über die Java ME und einem Vergleich dieser Java-Plattform mit der Java SE.

2.1 IT-Sicherheit

In diesem Kapitel sollen zunächst die für die weitere Arbeit notwendigen IT-Grundbegriffe eingeführt werden. Danach werden die notwendigen Schutzziele zur Umsetzung einer IT-Sicherheit definiert.

2.1.1 IT-Grundbegriffe

IT-Sicherheit bezieht sich auf die Sicherheit informationstechnischer Systeme. Deshalb erfolgt zunächst eine Definition des Begriffs IT-System.

Definition 2.1 *IT-System: Ein IT-System ist ein geschlossenes oder offenes bzw. verteiltes, dynamisches, technisches System mit der Fähigkeit zur Speicherung, Verarbeitung und Bereitstellung von Informationen [21].*

IT-Systeme erfüllen damit bestimmte Funktionen für deren Nutzer. Neben der Speicherung und Verarbeitung von Informationen gehört auch die Bereitstellung und Verteilung von Informationen zu den Aufgaben eines IT-Systems. Unter den Begriff IT-System fallen Kombinationen

aus Hardware und Software mit den unterschiedlichsten Anwendungsbereichen. Die Anwendungsbereiche können in Computer, Subsysteme und Komponenten sowie IT-Komplexe unterschieden werden. Zu Computern zählen sowohl Großrechner, Arbeitsplatzsysteme, PCs als auch Kleinst-Computer. Unter Subsystemen und Komponenten können Bausteine und Programme, Betriebssysteme sowie Anwendungssysteme und Netzkomponenten verstanden werden. Unter dem Begriff IT-Komplexe können Rechenzentren, Kommunikationsnetze sowie auch Telekommunikationsanlagen zusammengefasst werden.[17]

Geschlossene IT-Systeme basieren auf der Technologie eines einzigen Herstellers. Sie sind zu Konkurrenzprodukten nicht kompatibel und ihre Verwendung ist auf einen bestimmten Benutzerkreis und auf ein festgelegtes räumliches Gebiet beschränkt. In der heutigen offenen, dynamischen IT-Welt, in der immer größere Vernetzungen zwischen verschiedenen Anwendungen existieren, verlieren geschlossene IT-Systeme mehr und mehr an Bedeutung. Offene Systeme sind dagegen vernetzte und physisch verteilte Systeme, die mit anderen Systemen über Kommunikationsprotokolle interagieren. Für die Kommunikation werden allgemein anerkannte Kommunikationsstandards eingesetzt, und die Systeme beinhalten Schnittstellen die eine Kommunikation erlauben. Die Unterscheidung zwischen einem offenen und geschlossenen System ist sehr wichtig, da für offene Systeme erweiterte Maßnahmen zur Erreichung eines sicheren Systems ergriffen werden müssen.[21]

Komponenten eines IT-Systems In der Definition von IT-Systemen (2.1) wird der Begriff Information verwendet, jedoch nicht erklärt was Informationen genau sind. In der Literatur lässt sich keine eindeutige Definition des Begriffs „Information“ finden. Im Folgenden soll daher Information als ein Datum mit einer Bewertung und einem Ziel, eingebettet in einen Kontext, verstanden werden. Damit sind Informationen als Daten bzw. Datenobjekte mit einem erhöhten Wert anzusehen. Datenobjekte lassen sich in passive und aktive Objekte unterscheiden. Als passive Objekte werden Objekte mit der Fähigkeit Informationen zu speichern bezeichnet. Dazu zählen beispielsweise Informationsspeicher wie Dateien. Aktive Objekte wie beispielsweise Prozesse können zusätzlich Informationen verarbeiten.[21]

Um auf Datenobjekte zugreifen zu können, besitzen sie wohl definierte Schnittstellen. Diese Schnittstellen werden durch Operationen bzw. Methoden repräsentiert und erlauben so die Nutzung dieser Objekte. Die Benutzer und alle Objekte eines Systems die durch einen Benutzer aktiviert wurden, werden als Subjekte des Systems bezeichnet. Wenn Subjekte mit Objekten interagieren und ein Informationsfluss stattfindet, so wird dies als Zugriff auf die Information bezeichnet. Zur Gewährleistung von Sicherheitsaspekten werden Zugriffsrechte vergeben die genau festlegen, wer autorisiert ist auf ein Objekt zuzugreifen.[21]

Die Schnittstellen, die eine Interaktion zwischen Subjekten und Objekten ermöglichen, werden auch als Informationskanäle bezeichnet. Informationskanäle können in Speicherkanäle, legitime und verdeckte Kanäle unterschieden werden. Speicherkanäle sind Objekte, die von mehreren Subjekten gemeinsam genutzt werden. Legitime Kanäle sind vorgesehene Kanäle, die durch die Nutzung von Methoden und deren Parameter repräsentiert werden können. Verdeckte Kanäle (*covert channel*) sind eigentlich nicht für den Informationsaustausch vorgesehen, können jedoch dafür missbraucht werden. Verdeckte Kanäle können beispielsweise mit Hilfe der Steganografie (siehe Kapitel 2.6) aufgebaut werden. Die Steganografie erlaubt es zum Beispiel, innerhalb eines ausgetauschten Bildes andere Informationen zu verstecken.

Aufgrund der Definition eines IT-Systems und den vorgestellten Komponenten eines IT-Systems können nun weitere Grundbegriffe festgelegt werden. Zunächst erfolgt die Definition des Begriffs IT-Sicherheit, der in drei verschiedene Aspekte untergliedert werden kann (vgl. [21]).

Definition 2.2 *IT-Sicherheit*

Funktionssicherheit Um die Funktionssicherheit (*safety*) eines IT-Systems zu gewährleisten, müssen die spezifizierten Soll-Funktionalitäten mit den realisierten Ist-Funktionalitäten übereinstimmen. Dies bedeutet, dass das System keinen unvorhersehbaren Systemzustand erreichen darf, der beispielsweise durch Implementierungs-, Bedienfehler oder auch Missbrauch verursacht werden könnte.

Informationssicherheit Die Informationssicherheit (*security*) eines IT-Systems bedeutet, dass kein Systemzustand dazu führen kann, dass es zu einer unzulässigen Veränderung, Gewinnung oder einem Verlust von Information kommen kann. Um die Informationssicherheit zu gewährleisten, muss zuerst die Funktionssicherheit erfüllt sein.

Datensicherheit Die Datensicherheit (*protection*) eines Systems setzt voraus, dass die Funktionssicherheit gegeben ist. Ist dies erfüllt, muss zur Erreichung der Datensicherheit gewährleistet werden, dass kein Systemzustand dazuführen kann, einen unautorisierten Zugriff auf Systemressourcen und insbesondere die von dem System verarbeitenden Daten möglich zu machen. Dies bedeutet, dass keine Daten unautorisiert verändert, verloren oder auch hinzugefügt werden dürfen.
Die Funktionssicherheit eines IT-Systems ist sehr eng mit der Zuverlässigkeit bzw. Verlässlichkeit eines IT-Systems verwandt.

Definition 2.3 *Verlässlichkeit: Unter der Verlässlichkeit eines Systems ist die Eigenschaft eines Systems zu verstehen, keine unzulässigen Zustände anzunehmen (Funktionssicherheit) und zu gewährleisten, dass die spezifizierten Funktionen zuverlässig erbracht werden [21].*

Des Weiteren kann der Begriff IT-Sicherheit auch aus anderen Blickwinkeln betrachtet werden. Einerseits müssen IT-Systeme verlässlich sein. Dies bedeutet, dass sich Anwender eines IT-Systems auf dessen Korrektheit und Verfügbarkeit der Funktionen und der Ergebnisse dieser Funktionen verlassen können müssen. Andererseits müssen IT-Systeme auch beherrschbar sein. Das bedeutet, sie dürfen ihre Anwender weder direkt noch indirekt beeinträchtigen. Die Verlässlichkeit kann auch als „Sicherheit der Systeme“ und die Beherrschbarkeit als „Sicherheit vor dem System“ bezeichnet werden und stellt komplementäre Sichten des Begriffs IT-Sicherheit dar. Dierstein [18] spricht hier auch von der dualen Sicherheit.[18]
Abbildung 2.1 zeigt ein Schichtenmodell der IT-Sicherheit [18]. Hier sind die beschriebenen Sachverhalte und auch die im Folgenden vorgestellten Schutzziele bzw. Sicherheitsziele dargestellt.

2.1.2 Schutzziele

Informationen, die in einem IT-System gespeichert und verarbeitet werden, müssen vor den verschiedensten Bedrohungen geschützt werden. Um den Zugriff auf Informationen durch Subjekte zu sichern, sollte zunächst die *Authentizität* eines Subjekts festgestellt werden. Anhand der so festgestellten Identität kann nun ein Zugriff gestattet oder verweigert werden. Diese Zugriffskontrolle wird auch als *Autorisierung* bezeichnet. Wurde die Identität eines Subjekts erfolgreich festgestellt, so ist es möglich, die Schutzziele *Integrität* und *Vertraulichkeit* zu gewährleisten. Integrität und Vertraulichkeit sind sehr wichtige Schutzziele um beispielsweise vertrauensvolle elektronische Geschäftsbeziehungen über das Internet aufbauen zu können.

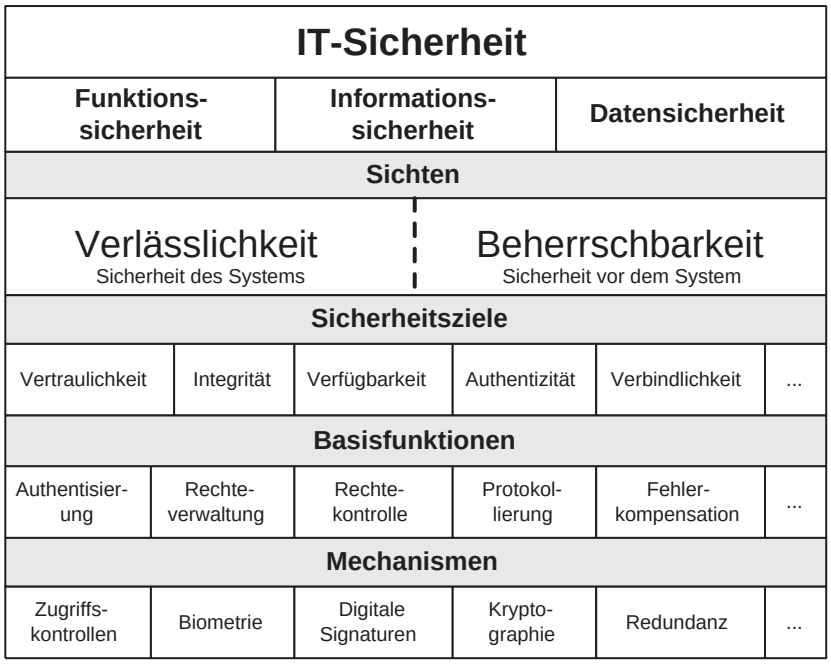

Abbildung 2.1: Schichtenmodell der IT-Sicherheit

Das System muss des Weiteren sicherstellen, dass auf die von dem Subjekt angeforderten Informationen auch zum Zeitpunkt der Anforderung zugegriffen werden kann. Gewährleistet das System diesen Sachverhalt, so wird das Schutzziel *Verfügbarkeit* erfüllt. Greift ein Subjekt auf eine Ressource zu, ist es vielfach wünschenswert, diesen Zugriff auch im Nachhinein eindeutig dem entsprechenden Subjekt zuzuordnen. Ist dies von dem System gewährleistet, wird das Schutzziel *Verbindlichkeit* bzw. *Zuordenbarkeit* erfüllt. Im Folgenden werden nun diese Schutzziele definiert und detaillierter vorgestellt.

2.1.2.1 Authentizität

Definition 2.4 *Authentizität: Unter der Authentizität (authenticity) eines Objekts bzw. Subjekts ist die Echtheit und Glaubwürdigkeit des Objekts bzw. Subjekts zu verstehen, die anhand einer eindeutigen Identität und charakteristischen Eigenschaft überprüfbar ist [21].*

Um die Authentizität einer von einem Nutzer vorgegebenen Identität zu überprüfen, existieren verschiedene Authentifikationsverfahren. Generell lassen sich diese in wissensbasierte- und besitzbasierte Verfahren unterteilen.
Wissensbasierte Verfahren nutzen dazu ein zwischen System und Subjekt zuvor vereinbartes Geheimnis, um die Authentizität festzustellen. Dazu fragt das System zunächst eine *Identifikationsbezeichnung (ID)* von dem Subjekt ab. Dadurch wird die Identität des Subjekts festgestellt. Existiert die Identität im System, so fragt das System das vereinbarte Geheimnis (*Challenge*) ab. Dies erfolgt meist durch die Abfrage eines Passworts (*Response*). Ist dieses korrekt, so ist die Authentizität des Subjekts festgestellt. Dieses Vorgehen wird auch als *Challenge Response-Verfahren (CR-Verfahren)* bezeichnet.
Besitzbasierte Verfahren setzen voraus, dass ein zu authentifizierendes Subjekt ein bestimmtes Merkmal besitzt. Dies können gegenständliche Merkmale wie beispielsweise eine Chipkarte

sein oder persönliche Merkmale. Persönliche Merkmale werden durch biometrische Verfahren festgestellt. Dadurch kann ein Fingerabdruck, die Stimme oder auch die Iris verwendet werden um die Authentizität festzustellen.
Beide Verfahren besitzen verschiedene Vor- und Nachteile. So muss bei wissensbasierten Verfahren sichergestellt werden, dass beispielsweise ein Passwort bestimmte Anforderungen erfüllt, um es gegen Angriffe durch raten zu schützen. Auch das Ausspähen von Passwörtern ist ein Nachteil der wissensbasierten Verfahren. Besitzbasierte Verfahren sind, wenn sie auf gegenständlichen Merkmalen basieren, besonders durch Diebstahl bedroht. Werden persönliche Merkmale verwendet, so bringt dies die Vorteile einer absoluten Eindeutigkeit und einer einfachen Handhabung. Außerdem sind persönliche Merkmale immer vorhanden und können nicht vergessen werden. Nachteile liegen in der *False Acceptance*, das heißt die falsche Akzeptierung eines Authentifikationsmerkmals und der *False Rejection*, bei der ein korrekter Benutzer nicht erkannt wird.
Um die Sicherheit der Authentifikation zu erhöhen, werden daher oft wissensbasierte- und besitzbasierte Verfahren kombiniert. Die so genannte *Zwei-Faktor Authentifikation* nutzt dabei die Vorteile beider Verfahren aus.

2.1.2.2 Integrität

Definition 2.5 *Integrität: Ein System gewährleistet die Datenintegrität (integrity), wenn es Subjekten nicht möglich ist, die zu schützenden Daten unautorisiert und unbemerkt zu manipulieren [21].*

Einerseits ist es zur Gewährleistung der Integrität notwendig, dass Subjekte nur die Daten verändern dürfen, für die sie ausreichende Rechte besitzen. Sie müssen also entsprechend autorisiert sein eine Veränderung vornehmen zu dürfen.
Zur Wahrung der Integrität muss andererseits festgestellt werden können, ob Daten unbemerkt manipuliert worden sind. Dazu werden Techniken, wie die elektronische Hash-Funktionen oder Signaturen eingesetzt, die in Kapitel 2.6 detailliert vorgestellt werden.

2.1.2.3 Vertraulichkeit

Definition 2.6 *Vertraulichkeit: Die Vertraulichkeit (confidentiality) von Informationen wird gewährleistet, wenn das System keine unautorisierte Informationsgewinnung ermöglicht [21].*

Wie bereits erwähnt, gehört die Vertraulichkeit zu den wichtigsten Schutzzielen um elektronische Geschäftsbeziehungen über das Internet abzuwickeln. Um die Vertraulichkeit einer Kommunikationsverbindung zwischen den Kommunikationsteilnehmern zu gewährleisten, wird im Wesentlichen die Technologie der Verschlüsselung eingesetzt. Dadurch können nur die Subjekte Einsicht in die ausgetauschten Daten nehmen, die einen entsprechenden Schlüssel zur Entschlüsselung besitzen. Zur Verschlüsselung existieren unterschiedliche Verfahren, die in Kapitel 2.6 ausführlich vorgestellt werden.

2.1.2.4 Verfügbarkeit

Definition 2.7 *Verfügbarkeit: Werden authentifizierte und autorisierte Subjekte in der Wahrnehmung ihrer Berechtigung nicht unautorisiert beeinträchtigt, so gewährleistet das System die Verfügbarkeit (availability) [21].*

Die Verfügbarkeit eines Systems ist ein Maß dafür, ob das System bestimmte Anforderungen zu einem Zeitpunkt bzw. innerhalb eines Zeitraums erfüllt. Die Verfügbarkeit wird auch als Verhältnis zwischen Gesamtlaufzeit weniger Gesamtausfallzeit und Gesamtlaufzeit ausgedrückt.
Durch die Gewährleistung der Verfügbarkeit, stehen einem Subjekt alle angeforderten Dienstleistungen, Funktionen oder Informationen eines IT-Systems zum gewünschten Zeitpunkt zur Verfügung. Wird ein System durch Angriffe in seiner Verfügbarkeit beeinträchtigt, so kann dies zu schwerwiegenden Wettbewerbsnachteilen für ein entsprechendes Unternehmen führen. Zur Gewährleistung der Verfügbarkeit müssen unterschiedliche Maßnahmen auf mehreren Ebenen ergriffen werden. Wichtigste Maßnahme ist die Verwendung von redundanten Komponenten. Beispielsweise sollten Zugangsleitungen zum Internet und eingesetzte Hardware redundant aufgebaut werden. Auch regelmäßige Datensicherungen (*Backups*) können das Maß der Verfügbarkeit eines Systems erhöhen.
Systeme lassen sich nach Verfügbarkeit klassifizieren. Angefangen von der einfachen Verfügbarkeit über Hochverfügbarkeit bis hin zur *Non Stop*-Verfügbarkeit müssen jeweils entsprechende Maßnahmen ergriffen werden, um die jeweiligen Anforderungen gewährleisten zu können.

2.1.2.5 Verbindlichkeit

Definition 2.8 *Verbindlichkeit: Die Verbindlichkeit bzw. Zuordenbarkeit (non repudiation, liability) einer Menge von Aktivitäten wird von einem System gewährleistet, wenn es nicht möglich ist, dass ein Subjekt im Nachhinein die Durchführung einer solchen Aktion abstreiten kann [21].*

Die Verbindlichkeit ist ein wichtiges Schutzziel, wenn Geschäfte elektronisch über das Internet abgewickelt werden. Dadurch kann erst die Rechtsverbindlichkeit einer durchgeführten Geschäftstransaktion garantiert werden. Unternehmen können sich somit darauf verlassen, dass beispielsweise eine getätigte Bestellung auch wirklich von dem vorgegebenen Kunden stammt und Kunden können sich sicher sein, dass das Unternehmen die Bestellung auch wirklich bekommen hat. Um die Verbindlichkeit zu gewährleisten, werden digitale Signaturen und Zertifikate eingesetzt, die in Kapitel 2.6 vorgestellt werden.
Zudem ist mit der Verbindlichkeit die Möglichkeit gegeben, eine Abrechnung der belegten Systemressourcen bzw. der genutzten Dienste vorzunehmen. Damit dies ermöglicht wird, muss jedoch zusätzlich auch eine Protokollierung einzelner Systemaktivitäten vorgenommen werden.

2.2 Bewertungskriterien

Die Einhaltung der in Kapitel 2.1.2 vorgestellten Schutzziele ist eine wesentliche Aufgabe um die Vertraulichkeit, Verfügbarkeit und Integrität von IT-Systemen zu gewährleisten. Verschiedene Kriterienkataloge sollen zur Beurteilung der Sicherheit informationstechnischer Systeme dienen. Unternehmen, die als Anwender solcher IT-Systeme auftreten, soll dadurch die Möglichkeit gegeben werden, entsprechende IT-Sicherheitsprodukte mit ähnlichem Funktionsumfang zu vergleichen und Passende auszuwählen. Zudem dienen die Kriterienkataloge dazu, den Entwicklern solcher Systeme eine Möglichkeit zu geben, gezielt sichere und vertrauenswürdige Systeme zu implementieren. Die Kriterienkataloge sollen des Weiteren eine objektive Bewertung durch externe und neutrale Instanzen erlauben und dadurch eine unabhängige Beurteilung der vom Hersteller versprochenen Sicherheitseigenschaften ermöglichen. Durch diese Instanzen ausgestellte Zertifikate sollen den Nutzern von zertifizierten Systemen die versprochene Sicherheit zusichern.

Die verschiedenen Kriterienkataloge, die zur Evaluierung der Sicherheitseigenschaften von IT-Systemen existieren, sind im Laufe der Jahre durch unterschiedliche Instanzen erarbeitet worden. Abbildung 2.2 zeigt die Kriterienkataloge und deren Abhängigkeiten bzw. Entstehungsgeschichte. Im Folgenden werden die wichtigsten dieser Kataloge vorgestellt. Zunächst werden die *Trusted Computer System Evaluation Critera* betrachtet, die die ersten derartigen Kriterien darstellen. Anschließend werden die in Deutschland entwickelten IT-Kriterien sowie die europäischen *Information Technology Security Evaluation Criteria* vorgestellt. Abschließend erfolgt die Betrachtung der *Common Criteria (CC)*, die einen international vereinheitlichten und weltweit anerkannten Kriterienkatalog darstellen.

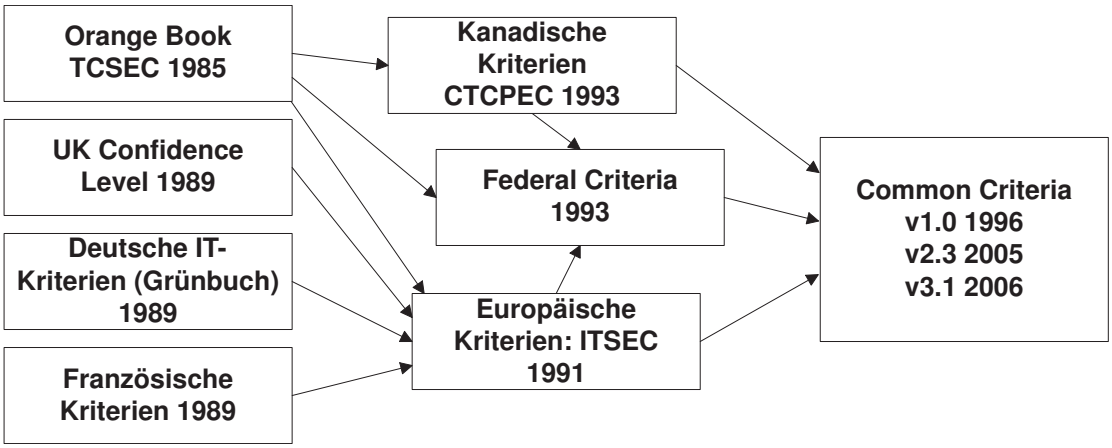

Abbildung 2.2: Abhängigkeiten zwischen IT-Kriterienkatalogen

2.2.1 TCSEC-Kriterien

Die *Trusted Computer System Evaluation Critera (TCSEC)* [76] sind die ältesten Kriterien zur Bewertung der Sicherheit von IT-Systemen, die im Jahr 1985 von dem *US National Computer Security Center* entwickelt wurden. Sie sind auch unter dem Namen Orange Book bekannt, was auf die Farbe des Katalogumschlages zurückzuführen ist.
Die TCSEC-Kriterien legen 7 Sicherheitsstufen zur Klassifikation der Sicherheit eines Systems fest. In Abbildung 2.3 sind diese hierarchisch aufgebauten Stufen neben den in den nächsten Kapiteln vorgestellten anderen Kriterien dargestellt.
Die TCSEC-Kriterien definieren den Bereich der alle Sicherheitsmechanismen eines Systems umfasst als *Trusted Computing Base (TCB)*. Die TCB wird auch als Sicherheitskern eines Systems bezeichnet und ist für die Durchsetzung der Sicherheitsrichtlinien zuständig. Alle Anforderungen, die in den verschiedenen Sicherheitsstufen der TCSEC-Kriterien genannt werden, sind weitgehend Anforderungen an die TCB.[76]
Die Sicherheitsstufe D stellt die niedrigste Stufe dar. In diese Stufen werden Systeme die keinen bzw. nur minimalen Schutz gewährleisten eingeordnet. Auf die Stufe D folgt die Stufe C, die sich in C1 und C2 untergliedert. Hierunter fallen Systeme die benutzerbestimmbaren Schutz bieten. Auf Stufe C1 muss eine Identifikation und Authentifikation von Benutzern erfolgen sowie eine Rechteverwaltung und Rechtekontrolle bzw. Zugriffskontrolle existieren. Die Stufe C2 erhöht die Anforderungen an die Rechteverwaltung. Hier müssen beispielsweise Zugriffsrechte auf einzelne Benutzer ausgedehnt werden und die Möglichkeit existieren, auch Zugriffsverbote

zu erteilen. Darüber hinaus fordert die Stufe C2 noch die Möglichkeit der Abrechenbarkeit von Benutzeraktionen sowie die Dokumentation und Auswertung von Überwachungsdaten.[21, 76]
Aufbauend auf der Stufe C2 existieren auf der nächsten Sicherheitsebene B die Stufen B1, B2 und B3. Systeme die in Stufe B eingeteilt werden, müssen systembestimmten Schutz bieten. Dies bedeutet, dass Sicherheitsstufen und Sicherheitsklassifikationen für Subjekte und Objekte existieren. Auf Stufe B1 werden zusätzlich zu den Anforderungen der Stufe C2 systembestimmte Zugriffsrestriktionen gewährleistet, die generell eine höhere Priorität als benutzerbestimmte Zugriffsrestriktionen besitzen. Zudem muss eine informelle Beschreibung der Sicherheitsmechanismen mit einer Begründung ihrer Wirksamkeit existieren.[76, 21]
Die Stufe B2 erfordert ein formales Sicherheitsmodell. Außerdem müssen Maßnahmen zur Erkennung von verdeckten Kanälen vorhanden sein und es muss möglich sein, einen vertrauenswürdigen Pfad (*trusted path*) zwischen TCP und Benutzer für die Identifikation und Authentifikation zu erstellen.[76]
Systeme, die in Stufe B3 eingeordnet werden, stellen erhöhte Anforderungen an den Sicherheitskern bzw. die TCB. Die TCB muss so klein wie möglich sein, um die Wahrscheinlichkeit einer fehlerhaften Implementierung zu minimieren und um eine formale Überprüfung zu vereinfachen. Die TCB muss getestet und gegen unbefugte Zugriffe geschützt sein. Außerdem müssen alle Sicherheitsaktivitäten der TCB überwacht werden.[21]
Letztlich folgt die Stufe A. Systeme dieser Stufe müssen keine weiteren funktionalen Erweiterungen als die der Stufe B3 erfüllen; jedoch müssen Systeme dieser Stufe formal beweisbar sein.
Die TCSEC-Kriterien waren eine erste Möglichkeit Systeme aus einem Sicherheitsblickwinkel zu evaluieren. Durch die einseitige Ausrichtung auf zentrale Betriebssysteme konnten jedoch andere Systeme nicht beurteilt werden. Zudem werden andere Schutzziele, wie Integrität und Verfügbarkeit, in den TCSEC-Kriterien eher ausgeklammert. Auch die Beschränkung auf die USA ist ein Kritikpunkt der TCSEC-Kriterien. Maßgeblich für die im Folgenden beschriebenen neueren Kriterienkatalogen war jedoch die fehlende Trennung zwischen der Bewertung der Sicherheitsfunktionalität und der Qualität, mit der diese Funktionalität erbracht wird.

2.2.2 IT-Kriterien

Die deutschen IT-Kriterien (*ITK*) bzw. IT-Sicherheitskriterien (*ITSK*) [6] wurden in den Jahren 1989/1990 von der damaligen *Zentralstelle für Sicherheit in der Informationstechnik (ZIS)* (heute *Bundesamt für Sicherheit in der Informationstechnik (BSI)*) erarbeitet und veröffentlicht. Die IT-Kriterien sind auch als Grünbuch bekannt und orientieren sich teilweise an den TCSEC-Kriterien, bringen jedoch auch viele Neuerungen mit sich.

2.2.2.1 Funktionsklassen

In den IT-Kriterien wurden Funktionsklassen zur Bewertung der Systemfunktionalität und Qualitätsstufen für die Bewertung der Qualität der eingesetzten Mechanismen eingeführt. Durch diese getrennte Bewertung wurde der größte Nachteil der TCSEC-Kriterien beseitigt.
In den Kriterien sind zehn Funktionsklassen beschrieben. Die ersten fünf Klassen entsprechen den Stufen, die in der TCSEC-Kriterien definiert sind. In Abbildung 2.3 wird dieser Zusammenhang grafisch verdeutlicht. Ab der Stufe sechs existiert keine hierarchische Ordnung dieser Stufen mehr, das heißt nächst höhere Stufen müssen nicht die Anforderungen ihrer vorherigen Stufen erfüllen. Systeme können jedoch durchaus die Anforderungen mehrere Funktionsklassen erfüllen.

TCSEC		ITSK	ITSEC
/		F10	F-DX
		F9	F-DC
		F8	F-DI
		F7	F-AV
		F6	F-IN
A		F5	F-B3
B	B3		
	B2	F4	F-B2
	B1	F3	F-B1
C	C2	F2	F-C2
	C1	F1	F-C1
D		/	/

Abbildung 2.3: Funktionsklassen der Kriterienkataloge TCSEC, ITSK und ITSEC

Die Klasse F6 stellt hohe Anforderungen an Systeme bezüglich der Integrität von Daten und Programmen, wie sie beispielsweise in Datenbanken erforderlich sein können. Neben der Identifikation und Authentifikation von Benutzern müssen Systeme dieser Kategorie Zugriffsrechte zwischen Benutzern, Rollen und Prozessen zu speziellen Objekten kennen und verwalten. Zusätzlich muss jeder Zugriffsversuch auf Berechtigung überprüft werden und eine erhöhte Beweissicherung durch eine Protokollierungskomponente vorgenommen werden. Zudem müssen alle Kommunikationen zwischen dem System und den Benutzern über einen vertrauenswürdigen Pfad erfolgen (vgl. [6] Seite 41 ff.).
Die Funktionsklasse F7 stellt hohe Anforderungen an die Verfügbarkeit eines Systems bzw. an die bereitgestellten Funktionen eines Systems. Dazu wird eine Fehlerüberbrückung gefordert, um die Aufrechterhaltung fortlaufend benötigter Funktionen zu gewährleisten. Des Weiteren muss das System für bestimmte festgelegte Funktionen eine maximale Reaktionszeit garantieren (vgl. [6] Seite 43).
Die Funktionsklassen F8 bis F10 beziehen sich auf die Übertragung von Daten. Dazu stellt Klasse F8 erhöhte Anforderungen an die Sicherung der Integrität von Daten während der Datenübertragung. So müssen Sender und Empfänger von Daten vor der Übertragung identifiziert, authentifiziert und diese Vorgänge protokolliert werden. Für die Sicherung der Übertragung müssen Fehlererkennungs- und Fehlerkorrekturmechanismen existieren. Darunter fallen auch Verfahren, die die Manipulationen des Datenstroms sowie erneut eingespielte Daten erkennen (vgl. [6] Seite 44 ff.).
Systeme, die eine hohe Anforderung an die Geheimhaltung von Daten während der Datenübertragung stellen, können in Klasse F9 eingeordnet werden. Hierzu zählen beispielsweise spezielle Verschlüsselungsgeräte (vgl. [6] Seite 46).
Die Funktionsklasse F10 umfasst sowohl Vertraulichkeitsanforderungen als auch Integritätsanforderungen vernetzter Systeme. Neben der Authentifikation der Kommunikationspartner wird eine Verschlüsselung sowie eine Protokollierung relevanter Systemaktivitäten gefordert.

2.2.2.2 Qualitätsstufen

Die in den ITSK definierten Qualitätsstufen dienen dazu, die Vertrauenswürdigkeit der vom Hersteller beschriebenen Funktionalität festzustellen. Die Qualitätsstufen berücksichtigen dazu

eine Reihe von Einzelaspekten. Hierzu gehören die Qualität der Sicherheitsanforderungen, die Qualität der Spezifikation der zu evaluierenden Systemteile sowie die Qualität der verwendeten Mechanismen. Zusätzlich wird die Qualität des Herstellungsvorgangs, die Betriebsqualität und die Qualität der Abgrenzung zu nicht zu evaluierenden Systemteilen bewertet. Auch die Qualität der Benutzerdokumentation gehört zu diesen Aspekten.
Die Bewertung der Qualität der verwendeten Mechanismen wird durch eine Skala festgelegt. Sie umfasst die Stufen „ungeeignet“, „schwach“, „mittelstark“, „stark“, „sehr stark“ und „nicht überwindbar“. Die Tabelle 2.1 zeigt die Charakterisierung dieser Stufen (vgl. [6] Seite 19 ff.).

Stufe	Charakterisierung
ungeeignet	Dieser Mechanismus bietet keinen Schutz und ist damit unwirksam.
schwach	Lediglich unbeabsichtigte Verstöße werden durch den Mechanismus verhindert. Absichtliche Verstöße können dagegen ohne große Hindernisse die Sicherheitsanforderungen verletzen.
mittelstark	Mechanismen dieser Stufe sind in der Lage, absichtliche Verstöße zu verhindern, bieten jedoch Personen mit normalen Kenntnissen über das System und mit mittelgroßem Aufwand die Möglichkeit, die Schutzmechanismen zur überwinden.
stark	Nur mit großem Aufwand und dem Einsatz von Hilfsmittel sind Mechanismen dieser Stufe zu überwinden.
sehr stark	Nur mit sehr großem Aufwand und dem Einsatz von aufwendigen Hilfsmitteln sind Mechanismen dieser Stufe zu überwinden.
nicht-überwindbar	Der eingesetzte Mechanismus stellt nach dem aktuellen Stand der Technik ein unüberwindbares Hindernis für absichtliche Verstöße dar.

Tabelle 2.1: Stufen der ITSK zur Bewertung der Stärke der Mechanismen

Die in den ITSK definierten acht Qualitätsstufen sind in Tabelle 2.2 zusammengefasst (vgl. [6] Seite 49 ff.). Die zweite Spalte zeigt dabei die Zuordnung der Stärke bzw. Wirksamkeit der eingesetzten Mechanismen zu diesen Qualitätsstufen.

Stufe	Wirksamkeit	Charakterisierung
Q0	ungeeignet	*unzureichende Qualität*: In diese Stufe werden Systeme eingeordnet, deren Qualität für keine der höheren Stufen ausreichend ist.
Q1	mittelstark (schwach in Ausnahmen)	*getestet*: Systeme dieser Stufen besitzen eine verbale Beschreibung der Sicherheitsanforderungen und die Spezifikation enthält eine grobe Beschreibung der Implementierung. Die Erfüllung der Sicherheitsanforderungen wurde mit einfachen Tests überprüft und es wurden dabei keine Fehler gefunden.
Q2	mittelstark	*methodisch getestet*: Systeme dieser Stufen besitzen eine verbale Beschreibung der Sicherheitsanforderungen, sind eingehend auf Korrektheit getestet und es wurden dabei keine Fehler gefunden.

Stufe	Wirksamkeit	Charakterisierung
Q3	stark	*methodisch getestet und teilanalysiert*: Systeme dieser Stufen besitzen eine detaillierte verbale Beschreibung der Sicherheitsanforderungen. Die Systemspezifikation wurde informell analysiert und auf ihre Konsistenz mit den Sicherheitsanforderungen überprüft. Kritische Stellen im Quellcode sind stichprobenartig analysiert. Es wurde methodisch getestet und dabei wurden keine Fehler festgestellt.
Q4	stark	*informell analysiert*: Entspricht der Stufe Q3, jedoch muss der Quelltext informell analysiert werden.
Q5	sehr stark	*semiformal analysiert*: Entspricht der Stufe Q4, jedoch muss der Quelltext semiformal analysiert werden. Zudem müssen die wichtigsten Sicherheitsanforderungen formal spezifiziert sein und die Systemspezifikation in semiformaler Notation vorliegen.
Q6	sehr stark	*formal analysiert*: Sicherheitsanforderungen und Systemspezifikation müssen zusätzlich in formaler Notation vorhanden und deren Konsistenz verifiziert sein. Zusätzlich muss der Quellcode auf Konsistenz mit der Spezifikation überprüft und das System methodisch getestet sein. Eine Fehlerfreiheit wurde festgestellt.
Q7	nicht-überwindbar	*formal verifiziert*: Entspricht der Stufe Q6. Der Quellcode muss formal auf Konsistenz mit der Spezifikation überprüft sein.

Tabelle 2.2: Qualitätsstufen der ITSK

Ist ein System anhand der IT-Kriterien untersucht worden, so kann ein Zertifikat ausgestellt werden. Dies enthält alle Funktionsklassen, die das System erfüllt sowie die erreichte Qualitätsstufe. Durch die Zertifizierung wird dem Hersteller eines Produkts das Vorhandensein der Sicherheitsfunktionalität bestätigt und dem Käufer eine Bewertungs- und Vergleichsmöglichkeit gegeben.

2.2.3 ITSEC

Die *Information Technology Security Evaluation Criteria (ITSEC)* [36] wurden im Jahr 1991 von der Europäischen Kommission veröffentlicht. Sie stellen eine europäische Vereinigung der bisherigen Kriterienkataloge der Länder Großbritannien, Frankreich, Niederlande und Deutschland dar.
Die ITSEC-Kriterien unterscheiden wie auch bereits die ITSK eine Bewertung der Systemfunktionalität und Qualitätsstufen für die Bewertung der Qualität der eingesetzten Mechanismen. Die Bewertung der Systemfunktionalität wird auch in den ITSEC-Kriterien anhand von Funktionsklassen vorgenommen. Diese sind analog zu den Funktionsklassen der ITSK aufgebaut, besitzen jedoch eine andere Namensgebung, die bei den ersten fünf Funktionsklassen ihre Ableitung von den TCSEC-Kriterien verdeutlicht (siehe Abbildung 2.3).
Die Bewertung der Qualität der Funktionen die das IT-System erbringt, wird in den ITSEC-Kriterien weiter aufgeteilt. Dabei wird die Bewertung der Funktionsweise des Evaluationsgegenstandes (des zu evaluierenden Systems bzw. Produkts) und die Bewertung der Wirksamkeit der eingesetzten Mechanismen getrennt vorgenommen. Die Wirksamkeit der eingesetzten Mechanismen wird im Gegensatz zu den ITSK nur anhand von drei Stufen vorgenommen. Die Stufen „niedrig“, „mittel“ und „hoch“ stellen damit ein Maß für das Vertrauen dar, inwieweit die Sicherheitsmechanismen in der Lage sind, direkten Angriffen zu widerstehen.[36]

Die Bewertung der Funktionsweise des *Evaluationsgegenstandes (EVG)* wird durch sieben Evaluationsstufen ermöglicht. Dadurch soll der Grad an Vertrauen ausgedrückt werden, der in die Korrektheit der Funktionalität des EVG gesetzt wird.[21, 36]
Für die Evaluation sind laut ITSEC verschiedene Sicherheitsvorgaben zu definieren. Dies umfasst eine Beschreibung der System-Sicherheitsrichtlinie bzw. des Produkts sowie eine Spezifikation der geforderten sicherheitsspezifischen Funktionen. Zudem die postulierte Mindeststärke der Mechanismen, die angestrebte Evaluationsstufe und optional eine Definition der geforderten Sicherheitsmechanismen. Für die Spezifikation der geforderten sicherheitsspezifischen Funktionen schlagen die ITSEC-Kriterien eine Einteilung anhand folgender Oberbegriffe vor. Diese sind mit den bereits in Kapitel 2.1.2 vorgestellten Schutzzielen verwandt und sollen das Verständnis und die Vergleichbarkeit der Sicherheitsvorgaben vereinfachen.

- Identifizierung und Authentifikation
- Zugriffskontrolle
- Beweissicherung
- Protokollauswertung
- Wiederaufbereitung
- Unverfälschtheit
- Zuverlässigkeit der Dienstleistung
- Übertragungssicherung

Die Evaluationsstufen beinhalten unterschiedliche Kriterien für die Bewertung der Korrektheit während der Konstruktion sowie während des Betriebs des EVG. Die Konstruktion wird unterteilt in eine Bewertung des Entwicklungsprozesses und der Entwicklungsumgebung. Der Betrieb wird in die Betriebsdokumentation und die Betriebsumgebung unterteilt. Jede dieser Bereiche ist durch unterschiedliche Phasen bzw. Aspekte gekennzeichnet. Für jede dieser Phasen bzw. Aspekte beschreiben die verschiedenen Evaluationsstufen Anforderungen an den Inhalt und die Form sowie an die zu erbringenden Nachweise und letztlich die Aufgaben des Evaluators. Abbildung 2.4 verdeutlicht diesen Zusammenhang.
In Abbildung 2.5 ist die Verwendung der Evaluationsstufen während des Entwicklungsprozesses aufgezeigt. Weiterführende Informationen dazu lassen sich in [36] ab Seite 45 finden.
Die Evaluationsstufen sind hierarchisch aufgebaut und beginnen bei der Stufe E0. In Stufe E0 werden Systeme eingeordnet, die keine oder nur unzureichende Vertrauenswürdigkeit bieten. Die Anforderungen an den EVG steigen von Stufe zu Stufe. Dies wird auch durch den Gebrauch der Verben „darlegen“ auf Stufe E1 und E2, „beschreiben“ auf Stufe E3 und E4 sowie letztlich „erklären“ auf den Stufen E5 und E6 verdeutlicht. Die Stufen E1 bis E6 werden wie folgt charakterisiert.

- E1: Die Stufe E1 fordert, dass eine informelle Beschreibung des Architekturentwurfs sowie die Sicherheitsvorgaben für den EVG vorliegen müssen. Die Sicherheitsvorgaben müssen die sicherheitsspezifischen Funktionen des EVG darlegen und zeigen wie diese Funktionen die Sicherheitsziele erfüllen. Die Beschreibung des Architekturentwurfs muss die Struktur und die Schnittstellen des EVG beinhalten. Funktionale Tests müssen nachweisen, dass der EVG die Anforderungen der Sicherheitsvorgaben erfüllt.

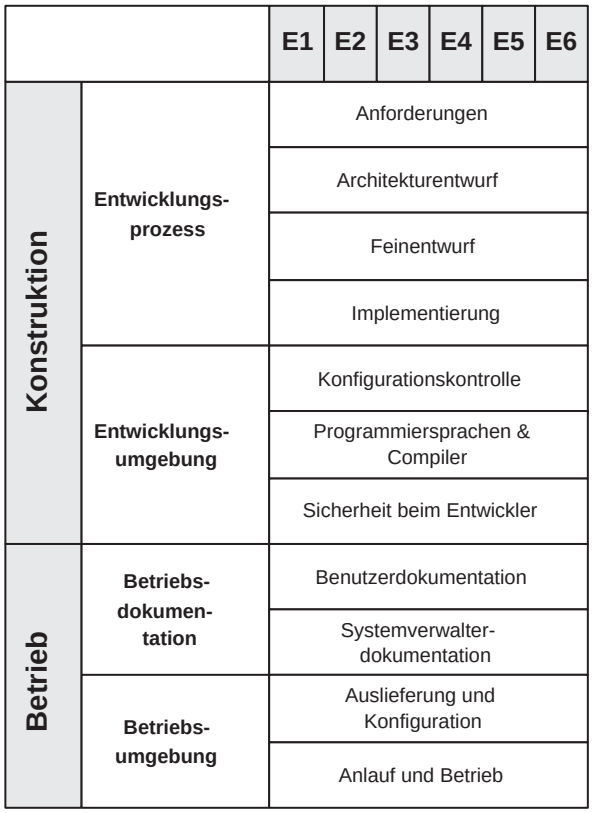

Abbildung 2.4: Gegenüberstellung der Evaluationsstufen zum Aufbau der Kriterien

- E2: Für System der Stufe E2 müssen alle Anforderungen der Stufe E1 erfüllt sein. Zusätzlich muss eine informelle Beschreibung des Feinentwurfs vorliegen. Weiterhin muss eine Testdokumentation vorliegen und die Bibliotheken der Testprogramme und -werkzeuge müssen für die Evaluierung verfügbar sein.
- E3: Zusätzlich zu den Anforderungen der Stufe E2 müssen der Quellcode beziehungsweise die Hardware-Konstruktionszeichnungen für alle sicherheitsspezifischen und sicherheitsrelevanten Systembereiche vorliegen. Eine informelle Zuordnungsbeschreibung, die den Bezug zwischen Quellcode bzw. Hardware-Konstruktionszeichnungen und dem Feinentwurf darstellt, ist ebenfalls gefordert.
- E4: Die Stufe E4 fordert die Erfüllung der Anforderungen der Stufe E3. Statt einer informellen Beschreibung der Architektur des EVG und des Feinentwurfs, fordert die Stufe E4 eine semiformale Beschreibung. Zusätzlich wird ein formales Sicherheitsmodell gefordert.
- E5: Die Anforderungen der Stufe E5 erfordern nun statt einer beschreibenden eine erklärende Form. Zusätzlich existieren noch weitere Anforderungen. Beispielsweise muss für den Implementierungsteil der Zusammenhang zwischen Feinentwurf und Quellcode erklärt werden und der Quellcode muss vollständig in kleine, verständliche und getrennte Teile aufgegliedert werden.
- E6: Die Stufe E6 fordert die Erfüllung der Anforderungen der Stufe E5. Zusätzlich müs-

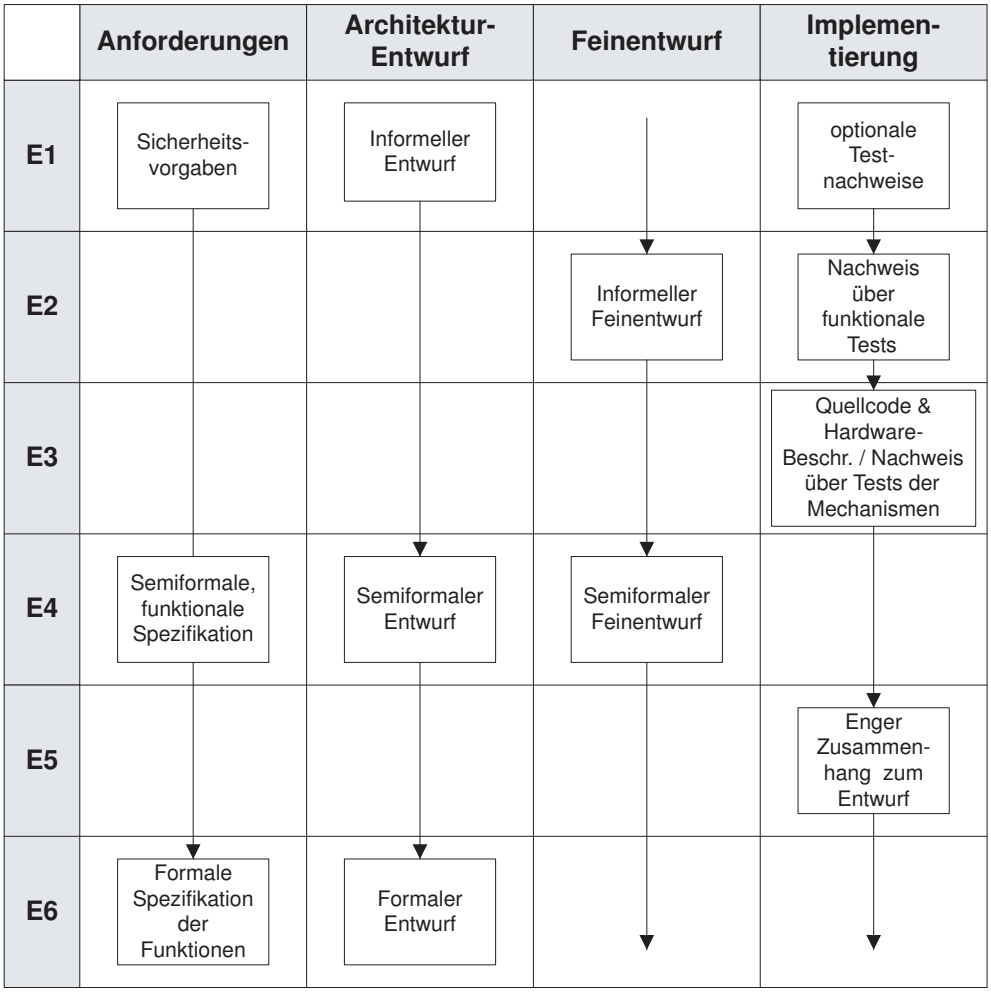

Abbildung 2.5: Korrektheitskriterien der Evaluationsstufen eines Entwicklungsprozesses

sen die sicherheitsspezifischen Funktionen und der Architekturentwurf in einer formalen Notation vorliegen. Diese muss konsistent mit dem zugrunde liegenden formalen Sicherheitsmodell sein.

Die Bewertung der Qualität der Funktionen, die das IT-System erbringt, wird durch eine Kombination aus Evaluationsstufe und der Einstufung der Wirksamkeit vorgenommen. Die Stufe (E3, mittel) beispielsweise bedeutet, dass mit zufriedenstellenden Mitteln die Korrektheit der Systemfunktionalität dargelegt und die Stärke der dafür eingesetzten Mechanismen mit „mittel“ bewertet wurde.[21]
Auf Basis der ITSEC-Kriterien kann vom Bundesamt der Sicherheit in der Informationstechnik (BSI) ein Zertifikat ausgestellt werden. Dazu muss eine technische Prüfung durch eine vom BSI anerkannte Prüfstelle erfolgen. Zertifikate können für fertige Produkte oder bereits zertifizierte Produkte ausgestellt werden. Nach erfolgreicher Zertifizierung erhält der Hersteller des EVG ein Zertifizierungsbericht.
Im Vergleich zu den ITSK und den TCSEC-Kriterien stellen die ITSEC-Kriterien einen Fortschritt dar. Jedoch ist eine zu starke Konzentration auf hierarchisch verwaltete Systeme weiterhin ein Manko. Dezentral organisierte, vernetzte IT-Systeme werden kaum berücksichtigt.

Ein weiterer Kritikpunkt ist auch die fehlende Berücksichtigung der unterschiedlichen Sicherheitsinteressen verschiedener Benutzergruppen. Des Weiteren wird davon ausgegangen, dass Bedrohungen nur von Nutzern des Systems oder Außenstehenden ausgehen. Gefahren, die etwa durch Betreiber oder Hersteller solcher Systeme existieren könnten, werden nicht beachtet. Die Durchführung eines Evaluationsprozesses ist außerdem sehr aufwendig und deren Ergebnis, der Zertifizierungsbericht, hat zu wenig Aussagekraft. So enthält ein Zertifizierungsbericht kaum nutzbare Informationen um eine Re-Zertifizierung eines weiterentwickelten Produkts zu vereinfachen. Die fehlende weltweite Anerkennung eines ITSEC-Zertifikats ist wohl einer der größten Nachteile in einer immer weiter fortschreitenden globalisierenden IT-Landschaft.

2.2.4 Common Criteria

Die *Common Criteria for Information Technology Security Evaluation (CC)* [81, 82, 83][1] stellen ein weiteres Kriterienwerk zur Prüfung und Bewertung der Sicherheit von IT-Systemen dar. Die CC sind ein internationaler Standard der eine weltweite Anerkennung finden soll. Zurzeit ist die CC in Europa, den USA, Kanada, Neuseeland, Australien, Japan und weiteren Ländern anerkannt und wird dort offiziell angewendet. Die Version v1.0 wurde im Jahr 1996 veröffentlicht und nach einer ausführlichen Testphase durch die Version v2.0 im Jahr 1998 ersetzt. Im Jahr 1999 wurde die Version v2.1 bekanntgegeben, die auch bei der Internationalen Standardisierungsorganisation (ISO) als Standard ISO 15408 veröffentlicht ist. Mittlerweile existiert die CC seit September 2006 in der Version v3.1, die in Kürze gemäß BSI-Zertifizierungsverordnung im Bundesanzeiger[2] offiziell veröffentlicht werden soll. Auch diese Version ist zur Standardisierung bei der ISO eingereicht worden. Zur CC gehört auch die gemeinsame Evaluationsmethodologie (*Common Methodology for Information Security Evaluation (CEM)*) [84], die eine abgestimmte Methodik für die Evaluierung auf Grundlage der CC festlegt. Die CEM beschreiben dabei wie eine Evaluation durchgeführt und die CC was evaluiert werden soll. Die CEM liegt ebenfalls in der Version v3.1 vor.
Die Version v3.1 bringt im Wesentlichen Änderungen im Bereich der Vertrauenswürdigkeitsanforderungen. Dazu wurden designspezifische Anforderungen überholt, die den ganzheitlichen Blick auf die Sicherheitsarchitektur eines Produkts verbessern sollen. Zudem sind Verbesserungen bei der Schwachstellenanalyse vorgenommen worden; auch die Anforderungen an die Produktentwicklung wurden harmonisiert. Konkret wurden dazu Aktivitäten, die nur einen geringen Beitrag zur Sicherheit des Systems beitragen, reduziert und ggf. eliminiert sowie redundante Evaluationsschritte entfernt. Um aufgetretene Missverständnisse besser vermeiden zu können, wurden außerdem einige CC-Terminologien klarer dargestellt. Des Weiteren wurden Informationen aufgenommen, die die Evaluierung von Systemen bzw. Produkten erleichtern sollen, die aus anderen bereits evaluierten Produkten aufgebaut sind (vgl. hierzu [83] Kapitel 9 - Composed Assurance Packages).
Wie bereits erwähnt, stellen die CC ein international vereinheitlichten und weltweit anerkannten Kriterienkatalog dar. Die CC entstanden im Laufe der Jahre durch eine Weiterentwicklung und Verschmelzung der bisherigen Kriterienkataloge. Abbildung 2.2 verdeutlicht diese Entwicklungsgeschichte der CC. Analog zu den ITSEC-Kriterien erfolgt bei den CC eine getrennte Bewertung der Funktionalität und der Vertrauenswürdigkeit. Die CC bietet jedoch detailliertere und konkretere Funktionalitätskriterien an.
In den CC werden die sieben Evaluierungsstufen EAL1 - EAL7 (*Evaluation Assurance Level*) für die Bewertung der Vertrauenswürdigkeit festgelegt. Diese sind fast analog zu den sechs

[1] http://www.commoncriteriaportal.org
[2] http://www.bundesanzeiger.de

Evaluierungsstufen der ITSEC-Kriterien aufgebaut. Ab der Stufe EAL2 entsprechen sie den ITSEC-Stufen. So entspricht die Evaluierungsstufe EAL2 der ITSEC-Stufe E1, die Stufe EAL3 der Stufe E2 bis hin zur Stufe EAL7, die der ITSEC-Stufe E6 entspricht. Die Stufe EAL1 wurde zusätzlich eingeführt um den Zugang zur Evaluierung zu vereinfachen bzw. zu erleichtern.
Die CC definiert ähnliche Vertrauenswürdigkeitskriterien wie die ITSEC. Beispielsweise wird die vierstufige Vertrauenswürdigkeitsbewertung des Entwicklungsprozesses in den ITSEC dargestellt durch „Anforderungen“, „Architekturentwurf“, „Feinentwurf“ und „Implementierung“ in den CC durch die „Funktionale Spezifikation des Sicherheitsmodells“, den „Entwurf auf hoher Ebene“, den „Entwurf auf niedriger Ebene“ sowie „Darstellung und Implementierung auf niedriger Ebene“. In Tabelle 2.3 (vgl. [7]) werden die Vertrauenswürdigkeitskriterien von CC und ITSEC gegenübergestellt. Es wird deutlich, dass sich alle Aspekte der ITSEC-Kriterien mit geringen Modifikationen auch in den CC wieder finden.

Common Criteria	**ITSEC**
Funktionale Spezifikation, Sicherheitsmodell	Anforderungen
Entwurf auf hoher Ebene	Architekturentwurf
Entwurf auf niedriger Ebene	Feinentwurf
Darstellung der Implementierung, Funktionale Tests	Implementierung
Konfigurationmanagement-Automatisierung, -Leistungsvermögen und -Anwendungsbereich	Konfigurationskontrolle
Werkzeuge und Techniken	Programmiersprachen und Compiler
Sicherheit bei der Entwicklung	Sicherheit beim Entwickler
Benutzerhandbuch	Benutzerdokumentation
Systemverwalterhandbuch	Systemverwalterdokumentation
Installation, Generierung und Anlauf	Anlauf und Betrieb
Auslieferung	Auslieferung und Konfiguration
Prüfung und Bewertung der Sicherheitsvorgaben	Eignung der Funktionalität
Prüfung und Bewertung der Sicherheitsvorgaben	Zusammenwirken der Funktionalität
Stärke der EVG-Sicherheitsfunktionen	Stärke der Mechanismen
Schwachstellenanalyse	Bewertung der Konstruktionsschwachstellen und der operationellen Schwachstellen
Missbrauch	Benutzerfreundlichkeit

Tabelle 2.3: Vertrauenswürdigkeitskriterien bei CC und ITSEC

Aufgrund dieser und weiterer Gemeinsamkeiten ist gewährleistet, dass bereits ausgestellte Zertifikate auf Basis der ITSEC-Kriterien ihre Gültigkeit in den CC wiederfinden.

Im Folgenden werden nun die einzelnen Teile der Common Criteria vorgestellt.

2.2.4.1 Einführung und allgemeines Modell

Der Teil 1 „Einführung und allgemeines Modell (*Introduction and General Model*)“ [81] der CC erläutert zunächst die Grundlagen der IT-Sicherheitsevaluation. Dazu werden beispielsweise Grundbegriffe wie Bedrohung, Sicherheitsziele und Sicherheitsanforderungen beschrieben und entsprechende Abkürzungen eingeführt. Danach werden wichtige Merkmale eines EVG bzw. TOE (*Target of Evaluation*) sowie die unterschiedlichen Zielgruppen der CC vorgestellt.
Die CC zielt auf drei unterschiedliche Benutzergruppen ab. Für Anwender stellt die CC Hintergrundinformationen bereit und bietet eine Anleitung für die Spezifikation von Anforderungen an die Funktionalität und Vertrauenswürdigkeit. Entwickler bekommen anhand der CC Hilfestellungen zur Formulierung von Sicherheitsspezifikationen, der funktionalen Spezifikation des EVG und der Anforderungen an die Vertrauenswürdigkeit des EVG. Die letzte Gruppe ist die der Evaluatoren. Sie bekommen anhand der CC Hilfestellungen um die Evaluation eines EVG vorzunehmen. Die Tabelle 2.4 (vgl. [81] Kapitel 6) beschreibt die Anwendung der einzelnen Teile der CC für diese drei Zielgruppen.

Teil der CC	Anwender	Entwickler	Evaluatoren
1	Hintergrundinformationen und zum nachschlagen. Allgemeine Anleitung zum Aufbau von PP.	Hintergrundinformationen und zum nachschlagen. Entwicklung von Sicherheitsspezifikationen für den EVG.	Hintergrundinformationen und zum nachschlagen. Allgemeine Anleitung zum Aufbau von PP und ST.
2	Hilfestellung und Referenz zur Formulierung von Anforderungen an den EVG	Referenz zur Interpretation von funktionalen Anforderungen und zur Formulierung von funktionalen Spezifikationen für den EVG.	Referenz zur Feststellung der Wirksamkeit der Sicherheitsfunktionen des EVG.
3	Hilfestellung zur Bestimmung der Vertrauenswürdigkeitsstufe des EVG	Referenz zur Interpretation von Anforderungen an die Vertrauenswürdigkeit des EVG und zu deren Formulierung.	Referenz zur Feststellung der Vertrauenswürdigkeit des EVG.

Tabelle 2.4: Anwendung der drei Teile der CC für die jeweiligen Zielgruppen

2.2.4.2 Schutzprofile

Der Nachteil der fehlenden Berücksichtigung der Sicherheitsinteressen verschiedener Benutzergruppen wird in den CC beseitigt. Hierzu werden im Anhang B des ersten Teils so genannte Schutzprofile (*Protection Profile (PP)*) eingeführt, die die unterschiedlichen Interessen und Anforderungen an einen EVG aus Anwendersicht beschreiben. Ein Schutzprofil enthält Anforderungen an die Funktionalität und die Vertrauenswürdigkeit für bestimmte Produkttypen bzw. Produktgruppen wie beispielsweise Firewalls, Datenbanken oder Chipkartenlesegeräte. Schutzprofile werden vor der eigentlichen Evaluierung erstellt und beschreiben unabhängig von der

zukünftigen Einsatzumgebung und von konkreten Produkten die Sicherheitsanforderungen an den EVG.
In Abbildung 2.6 (vgl.[81, 8]) ist der strukturelle Aufbau eines Schutzprofils dargestellt. Schutzprofile enthalten zunächst im Einführungsabschnitt eine eindeutige ID sowie einen kurzen Überblick und eine Beschreibung. Dadurch soll ein Anwender schnell entscheiden können, ob das Schutzprofil für ihn von Interesse ist. Der Abschnitt Konformität enthält Angaben zur Version der CC auf der das Schutzprofil aufbaut und ob evtl. eigene weitergehende Komponenten, auf der spezielle Sicherheitsanforderungen aufbauen, definiert sind. Eigene Komponenten können, falls sie nicht bereits in den Teilen 2 oder 3 der CC existieren, im letzten Abschnitt eines Schutzprofils eingeführt werden. Außerdem werden hier Bezüge bzw. Abhängigkeiten zu bereits bestehenden Schutzprofilen dargelegt. Im Weiteren werden in einem Schutzprofil die allgemeinen IT-Sicherheitseigenschaften sowie die Grenzen der Benutzung aufgezeigt. Neben dem Wert der Daten und deren Verarbeitung werden auch Annahmen an eine typische Einsatzumgebung festgelegt. Neben gesetzlichen Auflagen bzw. vorgeschriebenen Sicherheitsstandards werden außerdem alle abzuwehrenden Bedrohungen auf die zu schützenden Werte beschrieben. Zudem werden die zu schützenden Werte den Bedrohungen gegenübergestellt.[21, 8]
Die Sicherheitsziele beschreiben wie den genannten Bedrohungen entgegengewirkt werden soll. Dazu werden Sicherheitsziele für den EVG selbst und dessen Einsatzumgebung definiert. Durch die weitere Spezifikation von Anforderungen an die Sicherheitsfunktionalität und die Vertrauenswürdigkeit der Funktionalität wird die Basis für die Evaluierung und Zertifizierung gelegt. Die Anforderungen an die Funktionalität werden unter Anwendung der Funktionsklassen der CC (Teil 2) beschrieben. Die Anforderungen an die Vertrauenswürdigkeit sind in Form der bereits genannten Evaluierungsstufen vordefiniert und können zur Vergleichbarkeit genutzt werden.[21, 8]
Die CC definieren des Weiteren eine besondere Art von Schutzprofilen die nur die Evaluierungsstufe EAL1 erreichen können. Diese werden als „low assurance“ Schutzprofile bezeichnet. Zur Erstellung dieser Schutzprofile ist weniger Aufwand notwendig, da die Aspekte EVG-Problembeschreibung und Sicherheitsziele (in Abbildung 2.6 besonders gekennzeichnet) nicht benötigt werden.
Dieser Aufbau eines Schutzprofils soll den Verfasser eines Schutzprofils dazu zwingen, eine erste Konsistenz- und Vollständigkeitsanalyse durchzuführen und die Angemessenheit aller Anforderungen darzulegen.

2.2.4.3 Sicherheitsvorgaben

Soll nun ein konkretes Produkt oder System untersucht werden, so können die Sicherheitsanforderungen eines Schutzprofils in spezielle Sicherheitsvorgaben (*Security Target (ST)*) für den EVG überführt werden. Die Sicherheitsvorgaben haben einen ähnlichen Aufbau wie die Schutzprofile, jedoch werden detailliertere Angaben über die genaue Einsatzumgebung und den Gegenstand der Evaluierung gemacht. Ein ST bezieht sich damit immer auf genau eine Version eines konkreten Produkts eines Herstellers. Für den Fall, dass noch kein Schutzprofil für das zu evaluierende Produkt existiert, können die Sicherheitsvorgaben direkt formuliert werden.
Die Sicherheitsvorgaben werden vor der Evaluierung des EVG evaluiert, um festzustellen, ob die beschriebenen Bedrohungen, die geplante Einsatzumgebung, die Sicherheitsziele und Sicherheitsanforderungen zusammen passen. Dies soll sicherstellen, dass in den Sicherheitsvorgaben nur Anforderungen enthalten sind, die die Sicherheit auch wirklich erhöhen. Dadurch soll unnötige Komplexität von vornherein vermieden werden und so der Aufwand für die eigentliche Evaluierung des EVG minimiert werden.

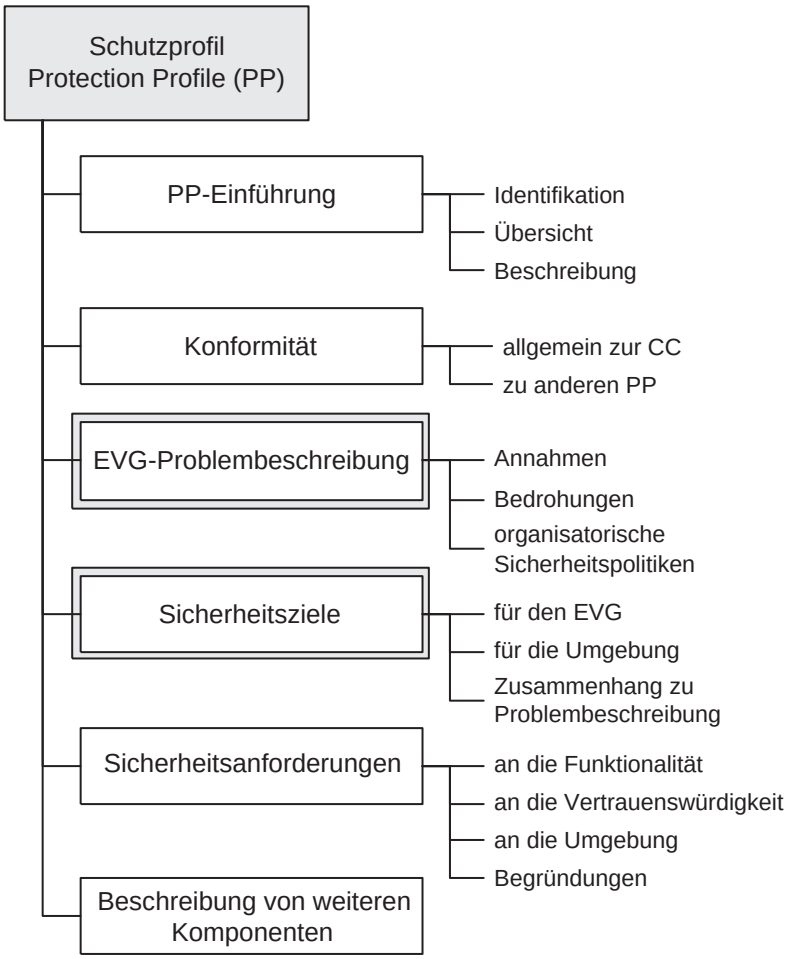

Abbildung 2.6: Struktureller Aufbau eines Schutzprofils

Die Teile 2 und 3 der CC beinhalten Kataloge von abgestimmten, evaluierbaren und in sich konsistenten Sicherheitsanforderungen an die Funktionalität (Teil 2) und die Vertrauenswürdigkeit (Teil 3). Diese dienen als Basis für die Spezifikation der Produkteigenschaften und die spätere Evaluierung und Zertifizierung.

2.2.4.4 Funktionale Sicherheitsanforderungen

Der Katalog der funktionalen Sicherheitsanforderungen (*Security Functional Requirements*) stellt ein empfohlenes Angebot für die Beschreibung der Anforderungen und Kriterien des Systems bzw. Produkts dar. Die Verwendung dieses Katalogs ermöglicht die spätere problemlose Evaluierung, da allgemein anerkannte Sicherheitsanforderungen an die Funktionalität verwendet werden. Diese beruhen auf den Erfahrungen aus der Verwendung von Vorversionen der CC sowie den bereits beschriebenen Kriterienkatalogen.

Die Sicherheitsanforderungen an die Funktionalität sind ähnlich zu den ITSEC-Funktionsklassen aufgebaut, sind jedoch insgesamt detaillierter. Zunächst wird in den CC eine Unterteilung in Klassen (Classes) vorgenommen. Die Klassen sind in Familien (Families) unterteilt, die selbst jeweils mindestens eine Komponente (Component) enthalten. Komponenten beschreiben kon-

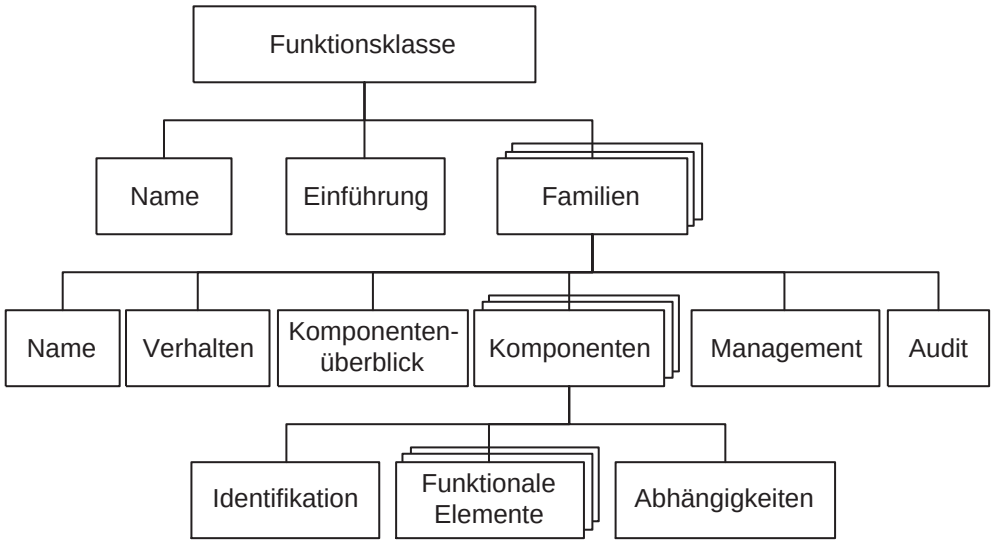

Abbildung 2.7: Aufbau der Funktionsklassen der Common Criteria

krete Anforderungen an die Funktionalität, wie beispielsweise an die Zugriffskontrolle oder die Beweissicherung.
Abbildung 2.7 zeigt den genauen Aufbau der Funktionsklassen. Die verschiedenen Funktionsklassen sind benannt und enthalten eine kurze Einführung die beschreibt, welchem Zweck die Klasse dient bzw. wie diese Klasse bestimmte Sicherheitsziele erfüllt. Zudem wird ein Überblick über den genauen hierarchischen Aufbau der Klassen gegeben. Die Familien enthalten neben dem Namen eine Beschreibung des Verhaltens der Familie, die die Sicherheitsziele und die funktionalen Anforderungen darstellt. Des Weiteren wird ein Überblick über die enthaltenen Komponenten gegeben sowie Anforderungen an das Management und die Überwachung festgelegt. Der Identifikationsteil der Komponente beinhaltet einen eindeutigen Namen und eine Einordnung in die Hierarchie der anderen Komponenten. Außerdem werden Abhängigkeiten aufgezeigt und die funktionalen Elemente aufgeführt. Funktionale Elemente sind die kleinsten Einheiten von Anforderungen an die Sicherheitsfunktionalität eines EVG.
Im Folgenden wird ein grober Überblick über die Funktionsklassen der CC gegeben [82, 21]. Nähere Informationen lassen sich im 2. Teil der CC [82] finden.

- **FAU**: Die Klasse *Functional Requirements Security Audit (FAU)* befasst sich mit der Sicherheitsprotokollierung, die das Erkennen, Aufzeichnen, Speichern und Analysieren von Informationen über sicherheitsrelevante Aktivitäten umfasst.

- **FCO**: Die Klasse *Functional Requirements Communication (FCO)* beschäftigt sich mit der Kommunikation. Dazu stellt sie zwei Familien bereit, die sich mit der Identifikation der am Datenaustausch beteiligten Instanzen befassen. Die Familien zielen darauf ab, den Urhebern von Informationen und deren Empfänger eindeutig zu identifizieren, um so Empfangsbeweise und Urheberschaftsbeweise zu ermöglichen.

- **FCS**: Die Klasse *Functional Requirements Cryptographic Support (FCS)* wird verwendet, wenn der EVG kryptografische Funktionen, die als Hardware, Firmware und/oder Software implementiert sein können, enthält. Diese werden zur Erfüllung von Sicherheitszielen, wie beispielsweise Identifikation, Authentifikation oder Verbindlichkeit, eingesetzt.

- **FDP**: Die Klasse *Functional Requirements User Data Protection (FDP)* wird für den Schutz der Benutzerdaten angewendet. Dazu sind vier Familien spezifiziert, die auf die Benutzerdaten in einem EVG während des Imports, Exports und der Speicherung sowie auf direkt mit Benutzerdaten verknüpften Sicherheitsattributen eingehen.

- **FIA**: Die Klasse *Functional Requirements Identification and Authentication (FIA)* beschäftigt sich mit der Identifikation und Authentifikation von Benutzern. Die Familien behandeln die Bestimmung und Verifizierung von Benutzeridentitäten, deren Berechtigung und die korrekte Zuordnung von Sicherheitsattributen.

- **FMT**: Die Klasse *Functional Requirements Security Management (FMT)* wird für das Sicherheitsmanagement verwendet. Neben der Festlegung verschiedener Managementrollen wird hier auch die Spezifikation des Managements von Sicherheitsattributen vorgenommen.

- **FPR**: Die Klasse *Functional Requirements Privacy (FPR)* enthält verschiedene Anforderungen zum Schutz der Privatsphäre. Dazu zählen Anforderungen zum Schutz der Enthüllung und des Missbrauchs von Benutzeridentitäten.

- **FPT**: Die Klasse *Functional Requirements Protection of the TSF (Target of Evaluation Security Functionality) (FPT)* dient zur Spezifikation von Anforderungen zum Schutz der Sicherheitsfunktionen des EVG. Dazu werden Anforderungen an die Integrität und das Management der Mechanismen, die die Sicherheitsfunktionen realisieren, gestellt.

- **FRU**: Die Klasse *Functional Requirements Resource Utilisation (FRU)* definiert Anforderungen an die Betriebsmittelnutzung. Hierbei wird besonders das Schutzziel Verfügbarkeit von Betriebsmitteln betrachtet. Außerdem existieren Familien, die auf die Fehlertoleranz eingehen. Dazu werden Schutzmittel gegen Nichtverfügbarkeit von bereitgestellten Funktionen definiert. Des Weiteren wird auf die Priorisierung von Diensten bzw. Aufgaben eingegangen.

- **FTA**: Die Klasse *Functional Requirements Target of Evaluation Access (FTA)* spezifiziert funktionale Anforderungen zur Kontrolle der Einrichtung einer Benutzersitzung. Hierbei wird beispielsweise auf Einschränkungen von mehreren gleichzeitigen Benutzersitzungen eingegangen. Außerdem werden Anforderungen an die für einen Benutzer einsehbare Historie von Zugriffen auf sein Benutzerkonto gestellt.

- **FTP**: Die Klasse *Functional Requirements Trusted Path/Channels (FTP)* stellt Anforderungen an einen vertrauenswürdigen Pfad zwischen einem Benutzer oder anderen Systemen und den Sicherheitsfunktionen des EVG.

2.2.4.5 Anforderungen an die Vertrauenswürdigkeit

Wie bereits erwähnt, werden die vorgestellten Sicherheitsvorgaben sowie die Schutzprofile vor der eigentlichen Evaluierung des EVG bereits separat evaluiert. Dazu enthält der dritte Teil „Anforderungen an die Vertrauenswürdigkeit (*Security Assurance Requirements*)“ [83] der CC nach einer Einführung die dafür notwendige Beschreibung der Kriterien.
Die Sicherheitsanforderungen an die Vertrauenswürdigkeit sind wie die Sicherheitsanforderungen an die Funktionalität durch Klassen, Familien und Komponenten strukturiert. Die Komponenten bestehen neben allgemeinen Elementen wie der Identifikation, Anwendungshinweisen

und Abhängigkeiten aus den Vertrauenswürdigkeitsanforderungen an den Entwickler, den Evaluator sowie an den Inhalt und die Form der Prüfnachweise. Jede Vertrauenswürdigkeitskomponente wird durch den Namen der Familie und einer Nummer identifiziert. Die Nummern werden innerhalb einer Familie pro Komponente sequentiell erhöht und spiegeln so die hierarchische Struktur dieser Komponente in der Familie wider.[83, 21]
In Tabelle 2.5 sind die unterschiedlichen Vertrauenswürdigkeitsklassen und deren Familien sowie Kurzformen im Überblick dargestellt. Nähere Informationen lassen sich in [83] ab Seite 77 finden.

Klassen	**Familien**	**Abkürzungen**
Klasse ADV: Entwicklung	Sicherheitsarchitektur	ADV_ARC
	Funktionale Spezifikation	ADV_FSP
	Darstellung der Implementierung	ADV_IMP
	Interna der EVG-Sicherheitsfunktionen	ADV_INT
	Sicherheitsmodell	ADV_SPM
	Designspezifikation	ADV_TDS
Klasse AGD: Handbücher	Anwenderhandbuch	AGD_OPE
	Installationsverfahren	AGD_PRE
Klasse ALC: Lebenszyklus-Unterstützung	Konfigurationsmanagementfähigkeiten	ALC_CMC
	Konfigurationsmanagement-anwendungsbereich	ALC_CMS
	Auslieferung	ALC_DEL
	Sicherheit bei der Entwicklung	ALC_DVS
	Fehlerbehebung	ALC_FLR
	Lebenszyklus-Beschreibung	ALC_LCD
	Werkzeuge und Techniken	ALC_TAT
Klasse ATE: Testen	Testabdeckung	ATE_COV
	Testtiefe	ATE_DPT
	Funktionale Tests	ATE_FUN
	Unabhängiges Testen	ATE_IND
Klasse AVA: Schwachstellen-bewertung	Schwachstellenanalyse	AVA_VAN
Klasse ACO: Komposition	Grundprinzip	ACO_COR
	Entwicklungsnachweise	ACO_DEV
	Vertrauen in abhängige Komponenten	ACO_REL
	Testen von zusammengesetzten EVG	ACO_CTT
	Schwachstellenanalyse von zusammengesetzten EVG	ACO_VUL

Tabelle 2.5: Vertrauenswürdigkeitsklassen und -familien

Die verschiedenen Vertrauenswürdigkeitsklassen und -familien bilden die Grundlage für die Festlegung der bereits genannten Evaluierungsstufen. In Tabelle 2.6 wird dieser Zusammenhang verdeutlicht. Die Vertrauenswürdigkeitsfamilien sind in den Zeilen und die Evaluierungsstufen in den Spalten dargestellt. Die daraus entstehende Matrix enthält jeweils Nummern, die eine der in der jeweiligen Familie definierten Vertrauenswürdigkeitskomponente identifiziert. Von Evaluierungsstufe zu Evaluierungsstufe werden Komponenten ggf. durch höherwertige ausge-

tauscht. Durch diesen Austausch wird dem höheren Anspruch an die Vertrauenswürdigkeit der jeweils nächsten Evaluierungsstufe Rechnung getragen.

Familie	Evaluierungsstufe EAL						
	1	**2**	**3**	**4**	**5**	**6**	**7**
ADV_ARC		1	1	1	1	1	1
ADV_FSP	1	2	3	4	5	5	6
ADV_IMP				1	1	2	2
ADV_INT					2	3	3
ADV_SPM						1	1
ADV_TDS		1	2	3	4	5	6
AGD_OPE	1	1	1	1	1	1	1
AGD_PRE	1	1	1	1	1	1	1
ALC_CMC	1	2	3	4	4	5	5
ALC_CMS	1	2	3	4	5	5	5
ALC_DEL		1	1	1	1	1	1
ALC_DVS			1	1	1	2	2
ALC_FLR							
ALC_LCD			1	1	1	1	2
ALC_TAT				1	2	3	3
ASE_CCL	1	1	1	1	1	1	1
ASE_ECD	1	1	1	1	1	1	1
ASE_INT	1	1	1	1	1	1	1
ASE_OBJ	1	2	2	2	2	2	2
ASE_REQ	1	2	2	2	2	2	2
ASE_SPD		1	1	1	1	1	1
ASE_TSS	1	1	1	1	1	1	1
ATE_COV		1	2	2	2	3	3
ATE_DPT			1	2	3	3	4
ATE_FUN		1	1	1	1	2	2
ATE_IND	1	2	2	2	2	2	3
AVA_VAN	1	2	2	3	4	5	5

Tabelle 2.6: Vertrauenswürdigkeitsfamilien und Evaluierungsstufen

2.3 Serviceorientierte Architekturen

Als Service- bzw. Dienst-orientierte Architekturen, kurz SOA (*Service Oriented Architecture*), werden Systemarchitekturkonzepte bezeichnet, die die Bereitstellung fachlicher Dienste und Funktionalitäten in Form von Services vorsehen. Systeme, die auf einer SOA basieren, werden dabei aus unterschiedlichen autonomen Diensten zusammengesetzt. Eine SOA verfolgt dabei drei wesentliche Ziele:

1. Die lose Kopplung von Softwaresystemen
2. Eine höhere Agilität von Geschäftsprozessen

3. Die verbesserte Wiederverwendung sowohl von Geschäftsprozessen als auch von Softwaresystemen

Das SOA-Konzept ist nicht völlig neu, sondern ist durch die Erfahrung beim Entwerfen und Entwickeln von verteilten Systemen entstanden. Es existieren schon seit den 80er Jahren Software-Architekturen, die das Prinzip einer serviceorientierten Architektur nutzen. Beispiele sind die von der *Object Management Group (OMG)*[3] entwickelte *Common Object Request Broker Architecture (CORBA)* [77] und das von Microsoft entwickelte *Distributed Component Object Model (DCOM)*[4]. Beide Konzepte nutzen Interfaces, um einem Nutzer den Zugriff auf die jeweiligen Funktionen zu ermöglichen. Die Nutzer und Anbieter einer Funktion können auf unterschiedlichen Systemen arbeiten. Die verwendete Architektur kümmert sich darum, dass ein Zugriff zwischen Anbieter und Nutzer möglichst einfach erfolgen kann. Dazu werden so genannte *Proxys* und *Stubs* verwendet, die jeweils den entfernten Kommunikationspartner repräsentieren. Die Verwendung dieser Konzepte bedeutet jedoch eine starke Kopplung bzw. Abhängigkeit zwischen den Kommunikationspartnern. Wird beispielsweise eine Schnittstelle verändert, so müssen auch die entsprechenden Softwarebestandteile, die diese Schnittstelle nutzen, bei den jeweiligen Kommunikationspartnern angepasst werden. Dies ist nur ein Grund, warum DCOM bzw. CORBA an Popularität verloren haben.
Durch was serviceorientierte Architekturen neuen Auftrieb bekommen haben, soll nun im Folgenden erläutert werden. Zunächst muss jedoch geklärt werden, was genau als Service bzw. Dienst bezeichnet wird und welche Merkmale eine serviceorientierte Architektur auszeichnen.

Definition 2.9 *Service oder Dienst: Ein Service bzw. Dienst ist eine eigenständige, zustandslose Funktion, die eine oder mehrere Anfragen bzw. eine oder mehrere Antworten über eine wohldefinierte Schnittstelle akzeptiert bzw. liefert. Services können einzelne Arbeitseinheiten ausführen und sind des Weiteren unabhängig von dem Zustand anderer Funktionen oder Prozesse.[39, 40]*

Im Umfeld einer serviceorientierten Architektur sind noch einige besondere Merkmale zu nennen. Services kapseln meist wieder verwendbare Geschäftsfunktionen ab und werden über Kommunikationsprotokolle, die die Interoperabilität und Ortstransparenz gewährleisten, dynamisch aufgerufen. Hinzu kommt, dass Services so implementiert werden, dass eine lose Kopplung zwischen Service-Anbieter und Service-Nutzer gewährleistet ist.

2.3.1 Interaktion in einer serviceorientierten Architektur

Der Service-Anbieter (*service provider*) implementiert einen Dienst und stellt ihn für den Zugriff durch Service-Nutzer im Netzwerk zur Verfügung. Dazu kann er in einem oder mehreren so genannten Service-Verzeichnissen Meta-Daten über seinen Dienst ablegen. Zu diesen Meta-Daten gehören beispielsweise Informationen über die Zugriffsmechanismen die der Service anbietet, eine genaue Beschreibung der Schnittstelle des Service, Informationen wo der Service im Netzwerk zu finden ist und evtl. auch Informationen über die Service-Provider selbst.
Kennt der Service-Nutzer die Schnittstelle und Adresse des Service, kann er diesen direkt nutzen. Ansonsten kann er über das Service-Verzeichnis diese Informationen abrufen. Danach kann der Zugriff auf den Service erfolgen. Dazu stellt er eine Anfrage (*service request*) an den Service und erhält nach der Bearbeitung eine Antwort (*service response*). Der Aufbau der Request- bzw. Response-Nachrichten ist durch die Schnittstellenbeschreibung genau festgelegt.

[3] http://www.omg.org
[4] http://www.microsoft.com/com/default.mspx

Das Service-Verzeichnis (*service-registry*) besitzt eine besondere Aufgabe in dieser Architektur. Das Verzeichnis ist eine Art Vermittler, der die Beschreibung eines Dienstes entgegennimmt und in einer Datenbank abspeichert. Zudem stellt es Abfragemöglichkeiten bereit, über den potenzielle Service-Nutzer gewünschte Dienste finden können. Ein Service-Verzeichnis ist in einer SOA nur ein optionales Element, das, falls sich Anbieter und Nutzer direkt kennen, nicht benötigt wird. Das Zusammenspiel dieser Rollen wird in Abbildung 2.8 noch einmal verdeutlicht.

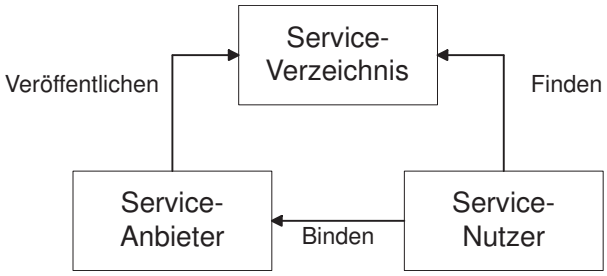

Abbildung 2.8: Die Rollen in einer serviceorientierten Architektur

Die beschriebenen Merkmale werden erst durch die Verwendung von Web Services (siehe Kapitel 2.4), zur Bereitstellung der Dienste einer serviceorientierten Architektur, vollständig erfüllt. In der Fachpresse und auf Tagungen wird deshalb die SOA auf Basis von Web Services als nächste große IT-Revolution bezeichnet. Damit soll eine neue Art der Entwicklung von Geschäftslösungen ermöglicht werden, die die Geschäftsprozesse im Vordergrund sieht und die Technik versteckt im Hintergrund lässt.

2.3.2 Merkmale von serviceorientierten Architekturen

Im Folgenden werden die Merkmale, die eine SOA auszeichnen, detailliert vorgestellt.

Dynamische Lokalisierung und Bindung Wie bereits beschrieben, nutzt der Service-Nutzer die Service-Beschreibung des Service-Anbieters, um eine korrekte Anfrage an den Service zu formulieren. Steht ein Service-Verzeichnis zur Verfügung, so kann der gewünschte Service dynamisch ausgesucht werden und dessen Lokalität anhand der Service-Beschreibung festgestellt werden. Die SOA erlaubt es nun, nach der dynamischen Lokalisierung auch eine dynamische Bindung zum Service herzustellen. Dazu kann anhand der Service-Beschreibung automatisch die Zugriffsschnittstelle auf Nutzerseite generiert werden. Zum Zeitpunkt der Kompilierung muss also nicht zwingend bekannt sein, welche Schnittstelle der Service besitzt und wie der Zugriff auf den Service erfolgt.

Wiederverwendung Ein wichtiges Merkmal einer SOA ist die Wiederverwendung. Die Wiederverwendung erlaubt es die Entwicklungskosten, den Entwicklungsaufwand und die Entwicklungsdauer zu minimieren. In dem bestehende Funktionen eines Systems als Services bereitgestellt werden, können diese in einem neuen Kontext wiederverwendet werden. So können beispielsweise auch bewährte Alt-Systeme (*Legacy-System*) einen neuen Stellenwert erhalten. Durch die Bereitstellung von Services, die es erlauben auf solche Systeme mit aktuellen Kommunikationsprotokollen zuzugreifen, können bereits getätigte Investitionen geschützt werden. Wiederverwendung kann auch bedeuten, dass bestehende Dienste intelligent zu neuen Diensten kombiniert werden.

Service-Komposition Die Service-Komposition beschreibt die Möglichkeit, neue Services aus vorhandenen aufzubauen. Durch eine sinnvolle Kombination von Services lassen sich so neue Funktionalitäten abbilden. Dazu ist es natürlich notwendig, dass entsprechend viele gut durchdachte Services existieren. Erst dann kann die Service-Komposition und damit auch die SOA ihren Vorteil ausspielen. Werden Services aus bestehenden Services zusammengesetzt, erhöht das zwangsläufig auch den benötigten Kommunikationsaufwand. Für die Komposition von Services spielt deshalb die Granularität der bereitgestellten Dienste eine entscheidende Rolle.

Granularität Bei der Umsetzung einer serviceorientierten Architektur stellt sich die Frage, was als Service bereitgestellt werden soll bzw. wie Services identifiziert werden. Technisch gesehen ist es möglich, jede Funktionalität als Service bereitzustellen. Jedoch kann dies zu einer sehr unübersichtlichen „Service-Welt“ führen. Deshalb muss bei der Definition von Services sehr umsichtig vorgegangen werden.

Damit nicht zu viele Services entstehen, ist es sinnvoll, Geschäftsfunktionalitäten als Service abzubilden. Wird dies nicht beachtet, könnte theoretisch jede Funktion als Service bereitgestellt werden. So könnte zum Beispiel auch eine Funktion, die lediglich den aktuellen Tag der Woche liefert, als Service bereitgestellt werden.

Daher muss die Granularität von Services genau überdacht werden. Werden Services zu feingranular definiert, entstehen generische Softwarebausteine mit geringem Funktionsumfang und hohem Wiederverwendungswert. Jedoch sinkt ihre tatsächliche Verteilbarkeit, da bei der Verwendung vieler solcher Services und Service-Stufen der Kommunikations-Overhead und damit der Kommunikationsaufwand steigt. Dies kann zu einer hohen Netzbelastung und damit ggf. auch zu Verzögerungen führen. Die Leistungsfähigkeit verschlechtert sich zunehmend mit der Anzahl der verwendeten Stufen.[9]

Grob-granulare Services sind dagegen für eine Verteilung besser geeignet. Die Choreografie zur Umsetzung der Prozesse wird außerdem vereinfacht. Der Grad der Wiederverwendbarkeit sinkt jedoch bei grob-granularen Services, da solche Services bereits umfangreiche Funktionalitäten bereitstellen.[9]

Lose Kopplung Die lose Kopplung erlaubt es, dass Services problemlos verändert, erweitert oder entfernt werden können, da keine bzw. nur minimale Anhängigkeiten zwischen den Service-Nutzern und Service-Anbietern existieren. Eine Kopplung kann auf verschiedenen Ebenen entstehen. Diese kann durch zeitliche, örtliche und technologische Abhängigkeiten entstehen. Ist beispielsweise ein zuvor genutzter Service in Netzwerk zurzeit nicht mehr verfügbar, so sucht sich ein Service-Nutzer einen anderen Anbieter mit der gewünschten Funktionalität. Die lose Kopplung erlaubt es nun diesen Service zu nutzen, auch wenn dieser sich an einer anderen Stelle befindet und dessen Schnittstelle nicht konform zum vorherigen Service ist. Wird das Konzept der losen Kopplung verwendet, so bedeutet dies, dass die Effizienz mit der Service-Anbieter und Service-Nutzer interagieren sinkt.

Ortstransparenz Das Konzept der losen Kopplung erlaubte es auch, dass der Ort, an dem ein Service erreichbar ist, nicht mehr zwingend zur Entwicklungszeit festgelegt werden muss. Stattdessen kann dieser problemlos verändert werden. Dies ermöglicht es, die Skalierbarkeit und Verfügbarkeit eines Service zu erhöhen. So kann auch besser auf Lastsituationen reagiert werden.

Interoperabilität Für den Begriff Interoperabilität existieren zahlreiche Erklärungen. Auf die IT bezogen bedeutet Interoperabilität die Fähigkeit der Zusammenarbeit von heterogenen Systemen oder Technologien. Die Interoperabilität ist in der SOA ein weiteres wichtiges Merkmal. Sie wird durch die Verwendung von bekannten und populären Standards erreicht, die es ermöglichen, den Zugriff auf einen Service völlig plattformneutral zu gestalten. Oft werden auch Adapter verwendet, die einem Service vorgeschaltet werden, damit dessen Funktionalität über unterschiedliche Zugriffsmechanismen und Zugriffsprotokolle erreichbar ist.
Im folgenden Kapitel wird die Web Service-Architektur vorgestellt, die maßgeblich zum neuen Erfolg der SOA beigetragen hat.

2.4 Web Services

Wie bereits erwähnt, können serviceorientierte Architekturen mittels unterschiedlicher Technologien realisiert werden. Durch das Aufkommen von Web Services hat diese Architektur jedoch einen neuen Auftrieb bekommen. Mittels Web Services lassen sich serviceorientierte Architekturen auf einfache Weise plattform-, programmiersprachen- und protokollunabhängig implementieren. Dies sind, neben der Modularität, Sicherheit und dem dezentralen Aufbau, die wichtigsten Anforderungen des *World Wide Web*-Konsortiums[5] (W3C) an die Web Services-Architektur.
Für den Begriff Web Services lassen sich zahlreiche Definitionen im Internet finden. Innerhalb dieser Arbeit soll die folgende Definition, in Anlehnung an die Definition der *W3C Web Services Architecture Group* [113], Verwendung finden:

Definition 2.10 *Web Service:*
Als Web Service wird eine durch einen Uniform Resource Identifier (URI) eindeutig identifizierbare Software-Anwendung bezeichnet, deren Schnittstellen in XML definiert, beschrieben und gefunden werden können. Ein Web Service unterstützt die direkte Interaktion mit anderen Software-Komponenten durch auf XML basierenden Nachrichten, die über Internet-Protokolle ausgetauscht werden.[38]

Für die Standardisierung der Web Services-Technologie existieren drei verschiedene Organisationen. Für die Kerntechnologie ist das W3C zuständig. Es kümmert sich zum Beispiel um die Entwicklung der *Extensible Markup Language (XML)* [116], der *Web Services Description Language (WSDL)* und *XML-Schema* [114]. Die *Organization for the Advancement of Structured Information Standards (OASIS)*[6] kümmert sich um Prozesse und die Komposition von Web Services. Dazu zählen beispielsweise Technologien wie *Electronic Business XML (ebXML)* [20], *Web Services Business Process Execution Language (WS-BPEL)* [3] sowie *Universal Description, Discovery and Integration (UDDI)* [11]. Die *Web Services Interoperability Organization (WS-I)*[7] definiert Profile und stellt Testumgebungen zur Verfügung, die die Interoperabilität von Web Services fördern soll.

Web Services-Architektur Die Web Services-Architektur orientiert sich sehr eng an dem Aufbau einer serviceorientierten Architektur (vgl. Kapitel 2.3). Innerhalb der Web Services-Architektur existieren auch drei verschiedene Rollen (vgl. Abbildung 2.9). Der Dienstanbieter

[5] http://www.w3c.org
[6] http://www.oasis-open.org
[7] http://www.ws-i.org

bzw. Service-Anbieter stellt mittels WSDL eine Beschreibung seiner Dienste (*services*) innerhalb eines Service-Verzeichnisses bereit. Dazu nutzt er Methoden von UDDI um seinen Dienst zu publizieren. Der Service-Nutzer kann wiederum mittels einer Methode von UDDI über das Service-Verzeichnis die Beschreibung eines Dienstes abrufen. Das Service-Verzeichnis stellt ein Verzeichnis der Dienste und unterschiedliche Schnittstellen für die Suche und das Publizieren von Diensten bereit. Hat ein Service-Nutzer eine WSDL-Beschreibung eines Dienstes abgerufen, kann er anhand der WSDL-Beschreibung manuell oder auch automatisch einen Client für den Zugriff erstellen. Die Kommunikation zwischen Service-Nutzer und Service-Anbieter erfolgt schließlich über das *Simple Object Access Protocol (SOAP)*.

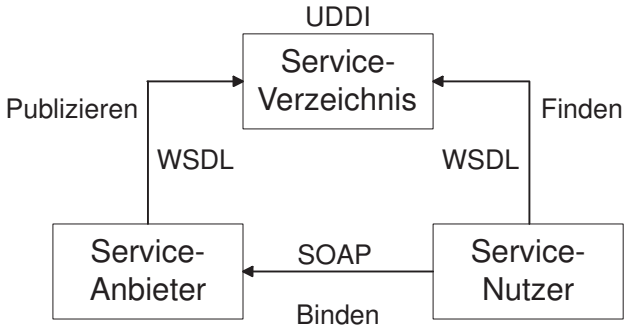

Abbildung 2.9: Web Services-Architektur

Web Services-Schichtenmodell Das Web Services-Schichtenmodell (siehe Abbildung 2.10) kann in Anlehnung an das *Open Systems Interconnection (OSI)*-Referenzmodell [13] verdeutlicht werden. Mittels der Entdeckungsschicht kann ein Service-Nutzer allgemeine Beschreibungen über den Dienst selbst und den Service-Anbieter einholen. Die Beschreibungsschicht dient dazu, dem Nutzer Angaben über mögliche Protokolle für den Zugriff, Informationen wo ein Service im Netzwerk zu finden ist sowie detaillierte Informationen über die Zugriffsschnittstellen bereitzustellen. Dazu zählt auch die Festlegung wie eine Anfrage (*request*) und eine Antwort (*response*) im jeweiligen Fall aussieht. Die Verpackungsschicht verpackt die Anwendungsdaten in ein XML-Dokument, das dann über die Transportschicht versendet werden kann. Die Transportschicht ermöglicht es, auf Basis der Netzwerkschicht unterschiedliche Zugriffsprotokolle bereitzustellen. Die Netzwerkschicht stellt Funktionalitäten für die Kommunikation, die Adressierung und das Routing bereit und entspricht damit der Netzwerkschicht des OSI-Modells.
Im Folgenden werden nun die einzelnen Technologien vorgestellt, die zum Erfolg der Web Services maßgeblich beigetragen haben.

2.4.1 XML

Die Basistechnologie, auf der alle relevanten Technologien im Umfeld von Web Services aufbauen, ist XML. XML ist trotz seines Namens keine Auszeichnungssprache (Markup-Sprache), sondern definiert stattdessen eine Menge von Regeln mit der Markup-Sprachen erstellt werden können. XML wurde aus der *Standard Generalized Markup Language (SGML)* [12] entwickelt, in dem spezifische Optionen und Abkürzungen aus SGML zur Vereinfachung der Syntax weggelassen wurden.

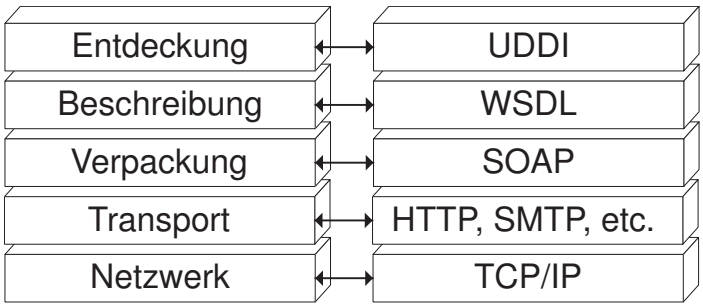

Abbildung 2.10: Web Services-Schichtenmodell

XML ist ein Standard mit dem Dokumente in Form einer hierarchischen Baumstruktur erstellt werden können. Aufgrund der Tatsache, dass XML-Dokumente sowohl die Information selbst als auch die Semantik der Informationsstruktur enthalten, sind XML-Dokumente sowohl von Menschen als auch von Maschinen lesbar. Abbildung 2.11 enthält ein einfaches Beispiel, das den logischen Aufbau von XML-Dokumenten verdeutlicht. Generell ist ein XML-Dokument in den Prolog (Abbildung 2.11, Zeilen 1-3) und die Baumstruktur der abzubildenden Daten aufgeteilt.

Die Baumstruktur des XML-Dokuments beginnt mit genau einem Element der so genannten Wurzel oder dem Dokumentelement. Ein Element wird durch einen Start-*Tag* (im Beispiel: `<bibliothek>`) begonnen und durch einen End-Tag (`</bibliothek>`) abgeschlossen. Ein Element enthält entweder direkt Text oder weitere Unterelemente, die auch als Kindelemente bezeichnet werden. Ein so genanntes Empty-Element-Tag (`<bibliothek/>`) ist ein spezielles Element das keinen weiteren Inhalt besitzt. Ein Element kann mehrere Attribute, wie in Abbildung 2.11, Zeile 6 dargestellt, enthalten. Attribute werden innerhalb eines Start-Tags durch einen Attributnamen gefolgt von einem „Gleichheitszeichen“ und in „Hochkommata“ dem Attribut-Wert gekennzeichnet. Des Weiteren können spezielle Verarbeitungsanweisungen (*processing instructions*) `<?Ziel-Name Parameter?>` sowie Kommentare `<!-- Kommentar-Text -->` innerhalb eines XML-Dokuments enthalten sein.

Der Prolog kann und sollte als erste Zeile die XML-Deklaration, das heißt Informationen über die verwendete XML-Version und den Zeichensatz, beinhalten. In Zeile 2 von Abbildung 2.11 ist ein Verweis auf eine *Document Typ Definition (DTD)* [117] enthalten. Die Angabe einer DTD ist optional und definiert die Grammatik eines XML-Dokuments und damit den genauen Aufbau des XML-Dokuments. Alternativ zu einer DTD kann auch eine *XML-Schema-Definition (XSD)* [114] verwendet werden, um die Grammatik eines XML-Dokuments zu definieren. Eine XSD ermöglicht es, die Grammatik eines XML-Dokuments noch genauer zu definieren. Beispielsweise können zusätzliche Angaben über Datentypen und deren Wertebereiche vorgenommen werden. Ein weiterer Vorteil einer XSD ist der dass, im Gegensatz zu einer DTD, eine XSD selbst auch wieder ein XML-Dokument darstellt. Zusätzlich kann der Prolog die Angabe eines *Stylesheets* enthalten. Ein Stylesheet ermöglicht es durch spezielle Regeln ein Layout für eine XML-Dokument festzulegen. Des Weiteren können mit Stylesheets auch Transformationen eines XML-Dokuments durchgeführt werden. So kann zum Beispiel ein XML-Dokument in ein *Hypertext Markup Language (HTML)*-Dokument [89] umgewandelt werden. Ein Stylesheet wird durch die *Extensible Stylesheet Langugage (XSL)* [118] definiert. Die XSL besteht aus den Teilen *XSL Transformations (XSLT)* [107], *XML Path Language (XPath)* [106] sowie *XSL Formatting Objects (XSL-FO)* [119].

```
 1:<?xml version="1.0" encoding="iso-8859-1"?>
 2:<!DOCTYPE bibliothek SYSTEM "bibliothek.dtd">
 3:<?xml-stylesheet href="liste.xsl" type="text/xsl"?>
 4:<!-- Ein Kommentar -->
 5:<bibliothek>
 6:   <buch genre="IT" isbn="3486578510" auflage="6">
 7:       <titel>IT-Sicherheit. Konzepte-Verfahren-Protokolle</titel>
 8:       <sprache>deutsch</sprache>
 9:       <autor geschlecht="w">Eckert, Claudia</autor>
10:       <inhalt>viel Text...</inhalt>
11:   </buch>
12:</bibliothek>
```

Abbildung 2.11: Aufbau von XML-Dokumenten

Damit ein XML-Dokument von einer XML-Anwendung interpretiert werden kann, muss es wohlgeformt (*well formed*) und ggf. auch gültig (*valid*) sein. Wohlgeformt bedeutet, dass ein XML-Dokument die syntaktischen Regeln der XML-Spezifikation erfüllt. Dazu zählt beispielsweise, dass ein End-Tag auf der gleichen Ebene verschachtelt ist, wie der zugehörige Start-Tag. Als gültig werden XML-Dokumente nur dann bezeichnet, wenn sie wohlgeformt sind und der Grammatik, die durch eine angegebene DTD bzw. XSD definiert wurde, entsprechen.
In der Praxis werden oft XML-Dokumente verschiedener Anwendungen kombiniert. Dadurch kann es zu so genannten Kollisionen kommen, wenn beispielsweise Elemente oder Attribute mit gleichem Namen in mehreren XML-Dokumenten existieren. Um dieses Problem zu beseitigen, existiert das Modell der Namensräume, das in Spezifikation [120] definiert ist. Mittels Namensräumen können Elemente und Attribute so erweitert werden, dass eindeutige Bezeichner entstehen und Kollisionen vermieden werden. Bei der Nutzung eines Namensraums wird ein Präfix deklariert, dass jedem Element das diesem Namensraum angehört, getrennt durch einen „Doppelpunkt“, vorausgestellt wird. In Abbildung 2.20 ist ein Beispiel für die Verwendung von Namensräumen dargestellt. Durch die Namensraum-Spezifikation können Elemente innerhalb eines XML-Dokuments durch einen so genannten *Qualified Name (QName)* eindeutig identifiziert werden. Ein QName besteht aus dem lokalen Bezeichner des Elements und einem vorangestellten Namensraumpräfix.

2.4.2 XML-Programmiermodelle

Um XML-Dokumente innerhalb von Anwendungen verwenden zu können, müssen sie zunächst eingelesen werden. Dadurch werden sie in die interne Datenstruktur der verwendeten Programmiersprache umgewandelt. Diese Umwandlung wird als *Parsen* und Werkzeuge die dies durchführen als *Parser* bezeichnet. Grundsätzlich werden zwei Modelle für das Parsen von XML-Dokumenten unterschieden. Beide Modelle besitzen unterschiedliche Eigenschaften und dadurch jeweils Vor- und Nachteile. Im Folgenden werden diese Modelle kurz vorgestellt. Es sei noch erwähnt, dass für diese Modelle jeweils zahlreiche Programmierschnittstellen in den unterschiedlichsten Programmiersprachen existieren.

2.4.2.1 Simple API for XML

Das Programmiermodell basierend auf der *Simple Application Programming Interface (AIP) for XML (SAX)*-Spezifikation [61] arbeitet Ereignis-gesteuert. Das bedeutet, dass die jeweilige

Anwendung bei jedem durch einen Parser erkannten Konstruktionselement entsprechend informiert wird. Dazu muss der Entwickler einen so genannten Ereignis-Handler implementieren, der alle Methoden enthält, die für die Bearbeitung des jeweiligen XML-Dokuments notwendig sind.

Nachdem der Ereignis-Handler bei einem Parser registriert wurde, werden die im Ereignis-Handler implementierten *Callback*-Methoden bei einem auftretenden Ereignis aufgerufen. So hat der Entwickler die Möglichkeit, auf einzelne Tags zu reagieren und kann so das Dokument bearbeiten. Abbildung 2.13 zeigt die beschriebene Vorgehensweise anhand eines Java-Programmierbeispiels, und Abbildung 2.12 stellt das Einlesen eines XML-Dokuments grafisch dar. Ein wichtiger Aspekt des SAX-Modells ist das sequentielle Einlesen des XML-Dokuments. Dadurch arbeiten SAX-Parser extrem schnell und besitzen einen sehr geringen Speicherbedarf. Die Bearbeitung von XML-Dokumenten mit Hilfe eines SAX-Parsers kann jedoch sehr komplex werden.

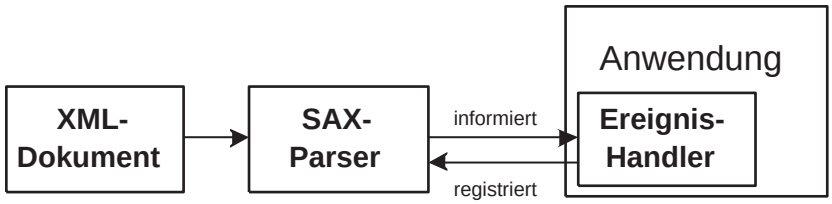

Abbildung 2.12: SAX-Übersicht

```
 1: java.io.File file= new java.io.File("beispiel.xml");
 2: java.io.FileInputStream fis= new java.io.FileInputStream(file);
 3: org.xml.sax.XMLReader saxReader;
 4: org.xml.sax.helpers.DefaultHandler handler;

 5: // SAX-Parser erstellen
 6: saxReader= org.xml.sax.helpers.XMLReaderFactory.createXMLReader();
 7: // Ereignis-Handler erstellen
 8: handler= new org.xml.sax.helpers.DefaultHandler() {
 9:   public void startElement(String uri, String localName,
10:     String qName, Attributes attributes) {
11:       System.out.println("Start-Element: " + localName);
12:   }
13: };

14: // Handler registrieren
15: saxReader.setContentHandler(handler);
16: // Starte das Parsen
17: saxReader.parse(new org.xml.sax.InputSource(fis));
```

Abbildung 2.13: Einlesen eine XML-Dokument mittels SAX-Parser

2.4.2.2 Document Object Model

Im Gegensatz zu SAX wird mit dem *Document Object Model (DOM)*-Programmiermodell [115] die baumbasierte Struktur eines XML-Dokuments auf ein Objektmodell der jeweiligen Programmiersprache abgebildet. Die Abbildungen 2.14 und 2.15 zeigen wie XML-Dokumente in Objektmodelle umgewandelt werden können bzw. ein Programmierbeispiel wie dies mit der Programmiersprache Java möglich ist. Das Objektmodell besteht aus unterschiedlichen Knoten (*nodes*), die verschiedenen Typen entsprechen und damit die unterschiedlichen Strukturelemente eines XML-Dokuments repräsentieren. Innerhalb der Programmiersprache Java repräsentiert der Knoten *Document* beispielsweise die Wurzel des XML-Dokuments. Für Attribute und Kommentare existieren die Knoten *Attr* bzw. *Comment*.
Mit Hilfe eines DOM-Parser können XML-Dokumente auf sehr einfache Weise bearbeitet werde, da, wie bereits erwähnt, das komplette Dokument verfügbar ist. Dadurch ergibt sich jedoch auch der Nachteil, dass DOM-Parser sehr speicherintensiv und langsam arbeiten. Dies wird besonders bei großen XML-Dokumenten deutlich.

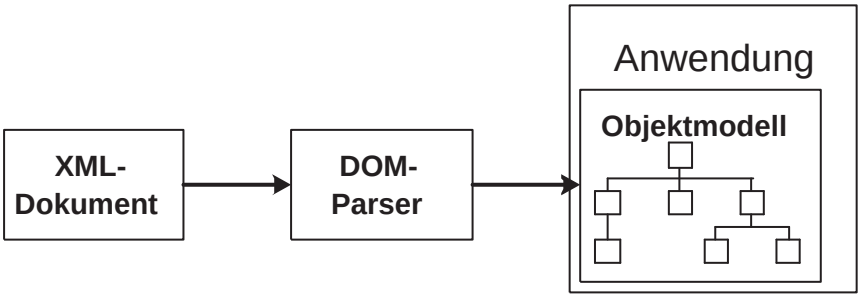

Abbildung 2.14: DOM-Übersicht

```
 1: java.io.File file= new java.io.File("beispiel.xml");
 2: java.io.FileInputStream fis= new java.io.FileInputStream(file);
 3: javax.xml.parsers.DocumentBuilderFactory factory;
 4: javax.xml.parsers.DocumentBuilder parser;

 5: // DOM-Parser erstellen
 6: factory= javax.xml.parsers.DocumentBuilderFactory.newInstance();
 7: parser= factory.newDocumentBuilder();

 8: // XML-Dokument erstellen
 9: org.w3c.dom.Document doc= parser.parse(fis);
10: System.out.println("1. Element: "
11:   + doc.getFirstChild().getLocalName());
```

Abbildung 2.15: Einlesen eine XML-Dokument mittels DOM-Parser

2.4.2.3 Streaming API for XML

Neben den vorgestellten Modellen hat sich ein neues Modell etabliert. Die *Streaming API for XML (StAX)* [10] ist ein *Application Programming Interface (API)*, eine Programmierschnittstel-

le, die zwischen SAX und DOM anzusiedeln ist. Wie bereits erwähnt, setzt DOM auf ein baumbasiertes Modell und SAX arbeitet Ereignis-gesteuert. Obwohl StAX eher dem SAX-Modell entspricht, verknüpft es doch beide Modelle und kombiniert dabei die einfache Nutzung von DOM mit der Effizienz von SAX. Im Gegensatz zu der Verwendung von SAX, wird die Anwendung nicht mittels der Erzeugung von Ereignissen über die unterschiedlichen Knoten eines XML-Dokuments informiert, sondern entscheidet selbst, welchen Knoten sie als nächstes für die weitere Verarbeitung benötigt. Die Kontrolle hat somit nicht der Parser sondern die Anwendung. Abbildung 2.16 zeigt ein Java-Programmierbeispiel, das die einfache Verwendung von StaX demonstriert. StaX unterscheidet zwischen einer Cursor-Verarbeitung (`XMLStreamReader`) und einer Iterator-Verarbeitung (`XMLEventReader`). Beide Verarbeitungsmodelle sind sich ähnlich, wobei sie sich im Wesentlichen dahingehend unterscheiden wie die Knoten des XML-Dokuments geliefert werden. Wird die Cursor-Verarbeitung verwendet, so werden die Knoten direkt mit dem Parser-Objekt (vgl. Abbildung 2.16 Zeilen 7,14 und 16) verarbeitet. Bei der Iterator-Verarbeitung liefert der erstellte Parser ein `XMLEvent`-Objekt, das somit für eine weitere Verarbeitung zur Verfügung steht.

```
 1: java.io.File file= new java.io.File("beispiel.xml");
 2: java.io.FileInputStream fis= new java.io.FileInputStream(file);
 3: javax.xml.stream.XMLInputFactory factory;
 4: javax.xml.stream.XMLStreamReader staxReader;

 5: // StAX Parser mit Cursor-Verarbeitung erstellen
 6: factory= javax.xml.stream.XMLInputFactory.newInstance();
 7: staxReader= factory.createXMLStreamReader(fis);

 8: for (int event= staxReader.next();
 9:   event!= javax.xml.stream.XMLStreamConstants.END_DOCUMENT;
10:   event= staxReader.next()) {
11:     switch (event) {
12:       case XMLStreamConstants.START_ELEMENT:
13:         System.out.println("Start-Element: "
14:           + staxReader.getLocalName());
15:         System.out.println("Element-Text: "
16:           + staxReader.getElementText());
17:       break;
18:       case XMLStreamConstants.END_ELEMENT:
19:         System.out.println("End-Element: "
20:           + staxReader.getLocalName());
21:       break;
22: }
```

Abbildung 2.16: Einlesen eines XML-Dokuments mittels StAX-Cursor-Verarbeitung

2.4.3 SOAP

Die Abkürzung SOAP [112] stand ursprünglich für „*Simple Object Access Protocol*“, dies wurde jedoch mit der Einführung der Version 1.2 verworfen[8]. Mittlerweile ist SOAP auch unter

[8]Die zugehörige Arbeitsgruppe war der Meinung, dass die Abkürzung SOAP die eigentliche Bedeutung des Protokolls nicht mehr eindeutig trifft.

dem Namen *XML Protocol (XMLP)* bekannt. SOAP dient als Kommunikationsprotokoll zwischen einem Web Service-Nutzer und einem Web Service-Anbieter. Dazu definiert die SOAP-Spezifikation ein standardisiertes Verpackungsprotokoll für den Nachrichtenaustausch zwischen Anwendungen. Wie auch alle anderen Spezifikationen im Web Services-Umfeld basiert SOAP auch auf XML. SOAP-Nachrichten sind protokollunabhängig und können daher über beliebige Transportprotokolle versendet werden. Das gängigste Transportprotokoll, das mit SOAP verwendet wird, ist das *Hypertext Transfer Protocol (HTTP)*-Protokoll [22]. Wie in Abbildung 2.17 zu sehen, ist jedoch auch die Verwendung von anderen Protokollen, wie beispielsweise das *Simple Mail Transfer Protocol (SMTP)* [28] oder das *File Transfer Protocol (FTP)* [24], möglich.

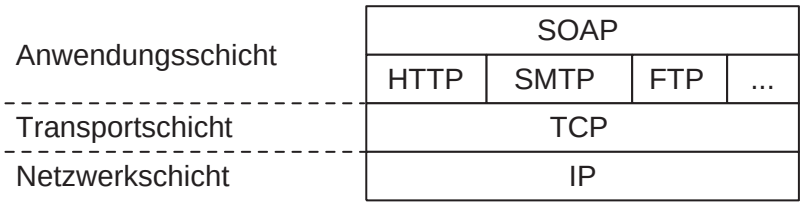

Abbildung 2.17: SOAP innerhalb des TCP/IP Protokollstapels

In Abbildung 2.18 ist die Struktur einer SOAP-Nachricht abgebildet. Der SOAP-*Envelope* stellt einen Container für mehrere SOAP-*Header* und genau einen SOAP-Body dar. Der SOAP-Header ist optional und dient zur Angabe von verschiedenen Informationen. Neben Verarbeitungs- und Routinginformationen können in einem SOAP-Header auch Angaben zu eventuell vorhandenen Verschlüsselungen und Signaturen des SOAP-Body eingetragen werden. Der SOAP-Body enthält die Nachrichten, die zwischen den beteiligten Instanzen ausgetauscht werden sollen.

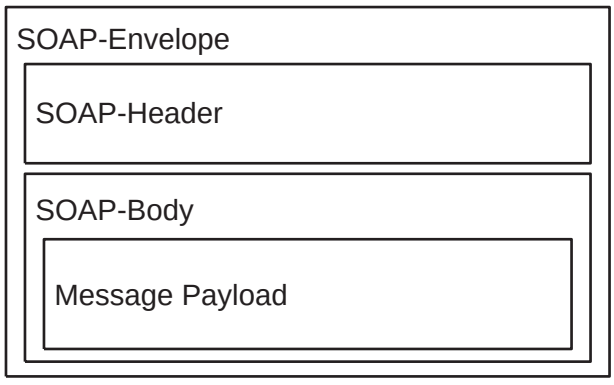

Abbildung 2.18: SOAP Nachrichtenaufbau

2.4.3.1 SOAP-Verarbeitungsmodell

Ein wichtiger Aspekt von SOAP ist das spezifizierte Verarbeitungsmodell (*SOAP processing model*). Eine SOAP-Nachricht muss nicht zwingend von einem Sender direkt zu einem Empfänger geleitet werden. Es kann durchaus sinnvoll sein, die Nachricht zunächst an verschiedene

Zwischenknoten zu senden. Diese modifizieren ggf. die Nachricht und leiten sie dann an den nächsten Knoten weiter. Der Weg einer SOAP-Nachricht wird auch als Nachrichtenpfad (*message path*) und die verschiedenen Instanzen, die eine SOAP-Nachricht senden, weiterleiten oder empfangen, als SOAP-Knoten (*node*) bezeichnet. Die SOAP-Spezifikation unterscheidet drei verschiedene Knoten. Der Sender, der eine Nachricht als erstes versendet, wird als „*initial sender*“ und der endgültige Empfänger einer Nachricht als „*ultimative receiver*“ bezeichnet. Die verschiedenen Zwischenknoten treten jeweils als Sender und Empfänger auf und werden daher als „*intermediary*“ bezeichnet.
Das SOAP-Verarbeitungsmodell erlaubt es für jeden SOAP-Knoten eine oder mehrere Rollen festzulegen, die der Knoten während der Bearbeitung einer Nachricht nicht verändern darf. Die Spezifikation definiert standardmäßig die Rollen `next`, `none` und `ultimateReceiver`. Erhält ein SOAP-Knoten eine Nachricht, wird zunächst eine Syntaxprüfung vorgenommen. Danach prüft er anhand der in den Header angegebenen Rollen und weiteren Attributen, ob er die entsprechenden Header und ggf. den Body bearbeiten muss. Nach der Bearbeitung wird die Nachricht weitergegeben bzw. bei einem Fehler der Sender informiert. Die SOAP-Spezifikation erlaubt es, neben den genannten Rollen auch benutzerdefinierte Rollen zu vergeben.

2.4.3.2 SOAP-Faults

Ein so genannter *SOAP-Fault* dient dazu, Fehlerinformationen innerhalb einer SOAP-Nachricht zu transportieren. Dazu definiert die SOAP-Spezifikation das Element `<env:Fault>`. Dieses wird bei einem Fehler als Element in den SOAP-Body eingefügt und an den Absender der fehlerhaften Nachricht gesendet.
Das Element `<env:Fault>` beinhaltet die zwei obligatorischen Kindelemente `<env:Code>` und `<env:Reason>` sowie optional anwendungsspezifische Informationen, die in das Kindelement `<env:Detail>` eingefügt werden können. Mittels des `<env:Code>`-Elements wird ein in der SOAP-Spezifikation definierter Fehlercode angegeben. Die Fehlercodes erlauben es, die Art des Fehlers genauer zu identifizieren. Die Spezifikation definiert dazu die Fehlercodes `VersionMismatch`, `MustUnderstand`, `DataEncodingUnknown`, `Sender`, `Receiver`. Für eine genaue Bedeutung dieser Fehlercodes sei hier auf das Kapitel 5.4.6 der SOAP-Spezifikation [112] verwiesen.
Das `<env:Reason>`-Kindelement enthält eine für den Menschen lesbare Form der Fehlerbeschreibung. Über die weiteren Kindelemente `<env:Node>` und `<env:Role>` kann mittels einer URI der SOAP-Knoten festgehalten werden, der den Fehler erzeugt hat bzw. die Rolle identifiziert werden, die der Knoten zum Zeitpunkt des Fehlers eingenommen hat. In Abbildung 2.19 ist die generelle Struktur einer *Fault*-Nachricht abgebildet.

2.4.4 WSDL

Die Popularität von Web Services wird maßgeblich durch die *Web Services Description Language (WSDL)* [122] beeinflusst. Die WSDL ist eine Beschreibungssprache, die es ermöglicht, die Schnittstellen eines Web Services unabhängig von Plattform, Programmiersprache und Protokoll einheitlich zu beschreiben. Mittels entsprechender Werkzeuge können somit automatisch Clients für den Zugriff auf einen Web Service erstellt werden, bzw. auch umgekehrt aus einer Java-Anwendung eine WSDL-Beschreibung generiert werden und so automatisch ein Web Service erzeugt werden. Dazu definiert WSDL auf einer hohen Abstraktionsebene, die Methoden die ein Web Service anbietet, deren Parameter sowie die Mechanismen und Nachrichtenformate, die für die Kommunikation mit dem Web Service verwendet werden können.

```
 1: <env:Envelope>
 2:    <env:Body>
 3:       <env:Fault>
 4:          <env:Code>...</env:Code>
 5:          <enc:Reason>...</env:Reason>
 6:          <env:Node>...</env:Node>
 7:          <env:Role>...</env:Role>
 8:          <env:Detail>...</env:Detail>
 9:       </env:Fault>
10:    </env:Body>
11: </env:Envelope>
```

Abbildung 2.19: Das SOAP-Fault Element

In Abbildung 2.20 ist der generelle Aufbau eines WSDL-Dokuments dargestellt. Es ist ersichtlich, dasd WSDL eine in XML definierte Sprache ist. Das WSDL-Dokument beginnt mit dem `<wsdl:definitions>`-Tag, in dem der Web Service-Name sowie verwendete Namensräume definiert sind. Das `<wsdl:message>`-Tag definiert die Nachrichten und deren Bestandteile, die verwendet werden müssen, wenn mit dem Web Service kommuniziert werden soll. Mittels des `<wsdl:portType>`-Tags werden der Methodenname sowie dessen Eingabe- und Rückgabe-Parameter festgelegt. Dazu enthält das Tag Verweise auf die bereits definierten `<wsdl:message>`-Tags. Das `<wsdl:binding>`-Tag dient dazu, die in dem Element `<wsdl:portType>` definierten abstrakten Methodennamen an ein konkretes Protokoll zu binden. Letztlich erfolgt mit dem `<wsdl:service>`-Tag die Angabe der Adresse, von der aus der Web Service erreichbar ist.

```
 1:<wsdl:definitions
 2:  name="ServiceName"
 3:  targetNameSpace="urn:Servicename"
 4:  xmlns:tns="urn:Servicename"
 5:  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
 6:  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl">

 8:   <wsdl:message name="ServiceInput"/>
 9:   <wsdl:message name="ServiceOutput"/>
10:   <wsdl:portType name="ServiceNamePortType"/>
11:   <wsdl:binding name="ServiceNameBinding"/>

12:   <wsdl:service name="ServiceName">
13:      <wsdl:port name="ServiceNameSoapPort"
14:        binding="tns:ServiceNameBinding">
15:         <soap:address location="http://www.service.de/service"/>
16:      </wsdl:port>
17:   </wsdl:service>
18:</wsdl:definitions>
```

Abbildung 2.20: Aufbau eines WSDL-Dokuments

2.4.5 UDDI

Unternehmen, die Web Services anbieten um für Kunden entsprechenden Dienste bereitzustellen, müssen dafür sorgen, dass ihre Dienste auch gefunden werden können. Um dies zu ermöglichen, musste eine Art Verzeichnis geschaffen werden, in dem Unternehmen Beschreibungen ihrer Dienste ablegen können. Dazu wurde Ende 2002 von IBM, Mircosoft und Ariba die Architektur *Universal Description, Discovery and Integration (UDDI)* [11] spezifiziert. UDDI ist vereinfacht ausgedrückt ein Verzeichnis (*registry*) für Dienste und deren Beschreibungen, das für die Suche und das Publizieren von Diensten entsprechende Schnittstellen anbietet.
UDDI gliedert sich dabei in drei verschiedene Verzeichnisse auf, die es ermöglichen sollen, auf unterschiedliche Weise schnell und effizient, einen Anbieter eines gewünschten Dienstes zu finden.

- Die *White Pages* enthalten den Namen, die Beschreibung und Kontaktinformationen eines Unternehmens, das Web Services anbietet.
- In den *Yellow Pages* können Web Services kategorisiert gefunden werden. Dazu werden international anerkannte Klassifizierungsstandards, wie beispielsweise die *United Nations Standard Products and Services Code (UNSPSC)*[9] Spezifikation, verwendet. Die Yellow Pages stellen damit ein Branchenbuch vergleichbar mit dem „Gelben Seiten“ dar.
- Die *Green Pages* enthalten schließlich die technischen Details der angebotenen Web Services. Dazu gehört als wichtigstes Element der Verweis auf die WSDL-Beschreibung eines Web Service.

Die UDDI-Spezifikation definiert für das Suchen und Publizieren von Diensten zwei Schnittstellen, die den Zugriff auf das Verzeichnis mittels des SOAP-Protokolls ermöglichen. Die Schnittstelle `InquireSOAP` ermöglicht anhand mehrerer Methoden das Finden eines Web Services sowie den Abruf von Details zu Web Services. Für das Publizieren, Ändern oder Löschen einer Web Services-Beschreibung wird `PublishSOAP` verwendet. Auch hier existieren mehrere verschiedene Methoden, mit dem Web Services bekannt gemacht werden können. Die genannten Schnittstellen sind selbst wieder als Web Services definiert und profitieren daher direkt von den Vorteilen der WS-Architektur.

2.5 Single Sign-On

Single Sign-On (SSO) bezeichnet ein Verfahren, das es einem Benutzer ermöglicht, nach einmaliger Anmeldung an einem besonderen Dienst auf alle angebotenen Dienste eines Netzwerks zuzugreifen, ohne dass hierfür jeweils eine erneute Anmeldung notwendig ist. Auf welche Dienste die Benutzer nach der Anmeldung durch den SSO-Authentifikationsmechanismus zugreifen dürfen, wird durch den Verbund an Diensten bestimmt, die der SSO-Infrastruktur angehören. Dem SSO kommt im Rahmen der serviceorientierten Architekturen eine besondere Bedeutung zu.
SSO vereinfacht die Authentifikation für den Benutzer erheblich. Ohne SSO muss sich ein Benutzer zahlreiche Benutzerkennungen und die zugehörigen Passwörter, die er für die Nutzung der unterschiedlichsten Dienstleistungen (E-Mail, Online-Shop, Foren, etc.) im Internet benötigt, merken. Mit SSO benötigt der Benutzer nur noch eine einzige Kennung für die angesprochenen Dienstleistungen. Der SSO-Mechanismus sorgt dafür, dass er bei jeder Dienstleistung

[9] `http://www.unspsc.org`

automatisch authentifiziert wird, sobald er sich mit dem SSO-Authentifikationsmechanismus angemeldet hat.
Ein SSO-System bringt sowohl für den Anwender als auch für die Betreiber des Systems viele weitere Vorteile.

- Einheitlicher Zugriff über einen einheitlichen Authentifikationsmechanismus.
- Die Anwender müssen sich nur noch eine Kennung und ein Passwort merken.
- Durch die Einmalanmeldung kommt es zu Zeitersparnissen.
- Erhöhung der Sicherheit durch die Verwendung von einem komplexen Passwort, statt vielen einfach zu merkenden Passwörtern.
- Der Benutzer muss sich pro Sitzung nur noch einmal manuell anmelden.
- Einsparpotenziale durch optimierte Wartbarkeit von Benutzer- und deren Zugangsdaten sowie durch Speicherung und Verwaltung dieser Daten in einer Datenbank bzw. durch einen Server.

Die genannten Vorteile von SSO überwiegen in den meisten Fällen die Nachteile. Die Nachteile dürfen jedoch bei der Implementierung eines SSO-Systems nicht vernachlässigt werden; ihnen muss stattdessen mit besonderen Anforderungen an das System begegnet werden.

- Kommt ein Angreifer in den Besitz der Zugangsdaten einer SSO-Kennung, so hat er direkten Zugriff auf alle Dienste der SSO-Infrastruktur.
- Der SSO-Authentifikationsmechanismus ist ein *Single Point of Failure*. Fällt dieser Mechanismus aus, wird die gesamte SSO-Infrastruktur gestört. Daher muss dieser Mechanismus besonders gegen Angriffe und Lastspitzen geschützt werden.

Im nächsten Abschnitt wird der *Kerberos*-Authentifikationsdienst vorgestellt. Das Kerberos-Protokoll ist das in der Praxis am häufigsten verwendete Protokoll um ein SSO zu realisieren. Danach wird kurz auf den von Microsoft mit dem Passport-Ansatz realisierten SSO-Mechanismus eingegangen. Die bekannteste Alternative zum Passportkonzept ist das Liberty-Projekt, das im Abschnitt 2.5.3 kurz vorgestellt wird. Im letzten Abschnitt dieses Kapitels wird noch die auf XML basierende *Security Assertion Markup Language* vorgestellt und gezeigt, wie damit ein SSO realisiert werden kann.

2.5.1 Kerberos

Kerberos (Versionen 1 bis 4) wurde ursprünglich am *Massachusetts Institute of Technology (MIT)* im Rahmen des Athena-Projekts[10] von Steve Miller and Clifford Neuman entwickelt. Es bietet ein Verfahren zur sicheren und einheitlichen Authentisierung von Benutzern in einem ungesicherten TCP/IP-Netzwerk aus sicheren Hostrechnern wie beispielsweise dem Internet. Der Name Kerberos stammt aus der griechischen Mythologie von einem gewaltigen Hund mit dem Namen Zerberus (lat. Cerberus), der den Eingang zur Unterwelt bewachte.
Kerberos liegt derzeit in der Version 5 [30] vor. Im Vergleich zur Version 4 haben sich viele Neuerungen ergeben und es wurden viele Fehler beseitigt. Nunmehr ist der verwendete symmetrische Verschlüsselungsalgorithmus nicht mehr nur auf DES festgelegt, sondern kann beim

[10] http://web.mit.edu/ist/topics/athena

Austausch der Nachrichten als Parameter angegeben werden. Die aktuelle Kerberos-Version unterstützt nun neben DES auch die Algorithmen 3DES, AES und RC4. Zudem ist es möglich, die Authentifikationsinformationen an andere Subjekte zu delegieren. Da zum Zeitpunkt der Entwicklung die Public Key-Verschlüsselung in den USA noch patentrechtlich geschützt war, unterstützt Kerberos derzeit nicht die Verwendung von asymmetrischen kryptografischen Systemen. In der nächsten Version soll dies jedoch berücksichtigt werden.
Für die Versionen 4 und 5 stehen sowohl freie als auch kommerzielle Implementierungen, jeweils für unterschiedliche Betriebssysteme, zur Verfügung. In aktuellen Windows-basierenden Netzwerken wird Kerberos als Standardprotokoll für die Authentifikation verwendet, teilweise jedoch mit proprietären Erweiterungen, die die Kompatibilität mit der Standardimplementierung verhindert.
Zu den Hauptaufgaben von Kerberos zählt einerseits die Bereitstellung von Hilfsmitteln zur Authentifikation von Subjekten und andererseits der Austausch von so genannten Sitzungsschlüsseln. Subjekte werden in Kerberos-Jargon als *Principals* bezeichnet. Kerberos basiert auf dem klassischen Client-Server-Prinzip. Ein Principal benötigt vor dem Zugriff auf einen Netzwerkdienst ein so genanntes *Ticket*. Das Ticket wird von einem vertrauenswürdigen Kerberos-Dienst ausgestellt und erlaubt es, einen gewünschten Netzwerkdienst ohne weitere Authentifikation zu verwenden.

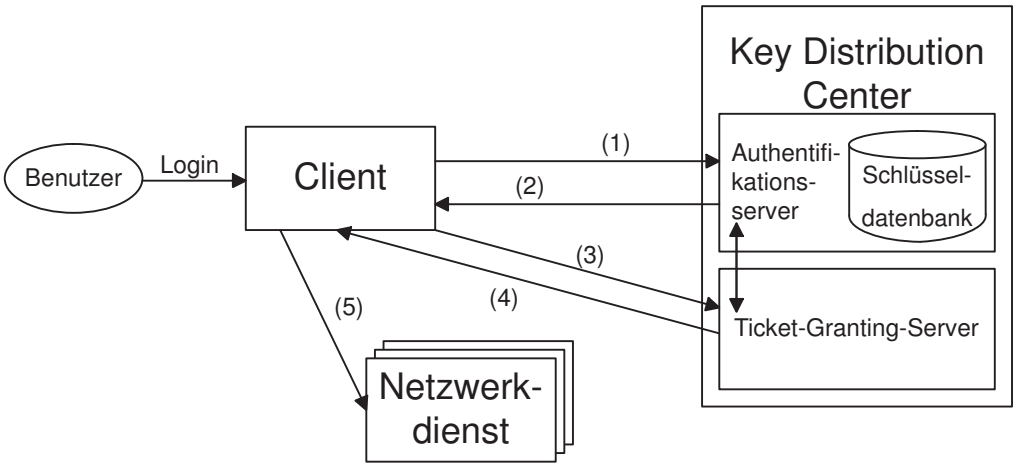

Abbildung 2.21: Kerberos Architektur und Protokollschritte

Protokollschritte Im Folgenden werden nun die einzelnen Protokollschritte detailliert vorgestellt (vgl. hierzu [30, 21]). Es sei noch darauf hingewiesen, dass die Nachrichten, die in den einzelnen Protokollschritten ausgetauscht werden, jeweils eine so genannte *Nonce* (*Number Used Nonce*) enthalten. Eine Nonce ist eine Zahl, die nur einmal verwendet wird. Sie wird eingesetzt um so genannte Replay-Attacken (vgl. Kapitel 5.1.4) abzuwehren.
Bei Kerberos wird zunächst eine Authentifikation am *Key Distribution Center (KDC)* vorgenommen. Das KDC besteht aus einem *Authentifikationserver* und einem *Ticket-Granting Server*. Der Authentifikationserver beinhaltet eine Datenbank mit den geheimen Schlüsseln der Benutzer und Netzwerkdienste des Systems. Die geheimen Schlüssel werden nicht im Klartext in der Datenbank abgespeichert. Zunächst werden Hash-Werte davon berechnet, die dann mit dem so genannten Master-Schlüssel des KDC chiffriert in der Datenbank abgelegt werden. Der Ticket-Granting Server ist für die Ausstellung der verschiedenen Tickets zuständig.

Im ersten Protokollschritt (vgl. Abbildung 2.21) startet der Principal eine Sitzung, in dem er sich mit seiner Benutzerkennung und seinem Passwort am Client anmeldet. Dazu sendet der Client eine Nachricht, bestehend aus seiner Benutzerkennung und einem chiffrierte Nachrichtenteil, an das KDC. Der chiffrierte Nachrichtenteil ist mit dem aus dem Benutzerpasswort abgeleiteten Schlüssel (dem Hash-Wert des Passworts) chiffriert und besteht neben Protokoll-spezifischen Parametern aus der Benutzerkennung sowie der aktuellen Uhrzeit.
Das KDC überprüft nun in seiner Datenbank, ob der Benutzer existiert und versucht dann, den chiffrierten Nachrichtenteil mit dem in der Datenbank gefundenen passenden Schlüssel zu dechiffrieren. Gelingt dies und ist der angegebene Zeitstempel nicht älter als 5 Minuten, so stellt er ein *Initialticket*, das so genannte *Ticket-Granting Ticket (TGT)* aus und verschlüsselt es mit seinem Master-Schlüssel. Das TGT enthält einen Sitzungsschlüssel, der mit dem Schlüssel des Benutzers chiffriert wird und zusammen mit dem TGT an den Benutzer gesendet wird (Protokollschritt 2).
Der Client kann nun mit seinem Passwort den Sitzungsschlüssel dechiffrieren und ihn zusammen mit dem TGT lokal abspeichern. Für eine sichere Aufbewahrung dieser Informationen ist der Client verantwortlich.
Will nun der Benutzer einen Netzwerkdienst in Anspruch nehmen, so stellt er eine erneute Anfrage an das KDC (Protokollschritt 3). Dazu sendet er das TGT, den Name des Service und einen so genannten Authentikator an das KDC. Der Authentikator ist eine Nachricht, die aus der Benutzerkennung und einem Zeitstempel besteht und mit dem gemeinsamen Sitzungsschlüssel chiffriert ist. Dadurch kann der Client seine Authentizität gegenüber dem KDC nachweisen und es wird verhindert, dass das TGT von einer anderen Person missbraucht werden kann. Das KDC entschlüsselt nun mit seinem Master-Schlüssel das TGT und erhält so den gemeinsamen Sitzungsschlüssel. Mit diesem Schlüssel kann das KDC nun den Authentikator dechiffrieren und so die Authentizität feststellen.
Ist die Authentizität erfolgreich festgestellt worden, so erstellt das KDC ein neues Ticket (*Folgeticket*) und einen neuen Sitzungsschlüssel für die Kommunikation zwischen dem Benutzer und dem angeforderten Netzwerkdienst und sendet diese Informationen in Protokollschritt 4 wieder an den Client.
Im letzten Protokollschritt sendet nun der Client einen mit dem gemeinsamen Sitzungsschlüssel des Benutzers und des Netzwerkdienstes erstellten Authentikator und dem zugehörigen Ticket an den Netzwerkdienst. Der Netzwerkdienst kann nun analog zu dem KDC die Authentizität des Benutzers nachweisen und so den Zugang gewährleisten oder ablehnen.
Die Authentifikation des Servers gegenüber des Clients kann mit Kerberos auch durchgeführt werden. Dazu muss der Server nur den Zeitstempel aus dem Authentikator des Clients um eins erhöhen und diesen mit dem gemeinsamen Sitzungsschlüssel chiffriert an den Client zurücksenden. Der Server weist dadurch seine Authentizität nach, da nur er den Sitzungsschlüssel dechiffrieren kann (mit seinem Master-Schlüssel) und damit Zugriff auf die Daten des Authentikators erhält (durch Entschlüsselung mit dem Sitzungsschlüssels).

Sicherheit Die Sicherheit von Kerberos ist mit der Version 5 erheblich verbessert worden, jedoch existieren weiterhin Sicherheitsprobleme (vgl. [21]).

- Die Verwaltung der Tickets und Sitzungsschlüssel auf Client-Seite ist problematisch, wenn der Client keine ausreichenden Sicherheitsvorkehrungen bietet.
- Auch das Passwortkonzept bietet Schwachstellen, wenn schwache Passwörter verwendet werden. Kommt ein Angreifer durch einen entsprechenden Angriff in den Besitz des Passworts, so kann er den gesamten Kerberos-Authentifikationsprozess unterlaufen.

- Um die Aktualität von Tickets und Anfragen zu überprüfen, enthalten die Nachrichten Zeitstempel. Diese werden auch für die Erkennung von Replay-Attacken verwendet. Um diese Sachverhalte erkennen zu können, sind alle beteiligten Instanzen der Kerberos-Infrastruktur über eine gemeinsame globale Uhrzeit synchronisiert. Ein Angreifer könnte nun die eingestellte Uhrzeit eines Servers verändern und die Wiedereinspielung von Nachrichten vornehmen, ohne dass dies der Server dann noch erkennen kann. Die Server der Kerberos-Infrastruktur sind also gegen solche Angriffe besonders abzusichern.

- Auch die Sitzungsschlüssel stellen einen Angriffspunkt dar. Sie sind meist für eine bestimmte Zeitspanne gültig, die in der Version 5 bis zu 21 Stunden betragen kann. Kommt ein Angreifer in den Besitz eines Sitzungsschlüssels der noch gültig ist, kann er diesen missbrauchen. Ein Schlüssel sollte daher nur solange gültig sein als unbedingt notwendig und ggf. auch zurückgerufen bzw. erneuert werden. Die Version 5 bietet dafür entsprechende Möglichkeiten an.

2.5.2 Microsoft Passport

Das Passport-Konzept [62, 63] ist Bestandteil der *Microsoft .NET* Strategie und beschreibt ein weiteres Verfahren, um den SSO im Internet zu ermöglichen. Durch die Verwendung des Passport-Konzepts erhält der Benutzer eine Online-Identität, eine so genannte *Windows Live-ID*. Dazu muss er sich zunächst bei einem von Microsoft angebotenen Dienst, wie beispielsweise dem E-Mail-Service *Hotmail*[11] oder dem Windows XP Registrierungsservice, registrieren. Die Online-Identität ermöglicht es einem Benutzer, nach erfolgreicher Authentifikation am Passport-Dienst, auf alle Internetseiten von Anbietern zuzugreifen, die sich für das Passport-Konzept registriert haben, ohne dass er sich hierfür erneut authentifizieren muss.
Die Zugangsdaten, die für eine Authentifikation benötigt werden, werden in einer zentralen Datenbank in der Domäne `.passport.com` gespeichert. Die Zugangsdaten bestehen aus einer 64-Bit langen so genannten *.NET Passport Unique ID (PUID)* sowie einem zugehörigen Passwort und sind zusammen mit einem Benutzerprofil verschlüsselt in der Datenbank abgelegt. Das Benutzerprofil kann optional noch Informationen wie das Alter, das Geschlecht, die Adresse oder die Sprache enthalten. Zusätzlich ist es auch möglich, so genannte *Wallet*-Daten mit im Profil abzulegen. Wallet-Daten beinhalten Informationen, die für Bestellungen und Zahlungen im Internet benötigt werden. Dies sind beispielsweise die Kreditkartennummer und die Lieferadresse eines Benutzers.
Mit dem beschriebenen Benutzerprofil soll das Passport-Konzept das Online-Shopping vereinfachen. Nun kann ein Benutzer direkt Bestellungen in einem Web-Shop aufgeben ohne sich dafür extra authentifizieren zu müssen. Die benötigten Zahlungsinformationen kann der Anbieter eines Shops direkt vom Passport-Server abrufen.

Passport-Modell Passport ist ein Web-basierter Dienst, der auf der Verwendung von Standardtechnologien wie *Secure Socket Layer (SSL)*, HTTP-Redirect [22], *Cookies* und *JavaScript* basiert. Zur Nutzung des Dienstes muss daher keine besondere Installation vorgenommen werden, setzt jedoch einen Browser voraus, der Cookies unterstützt und deren Verwendung aktiviert ist.[62]
Das Passport-Modell unterscheidet zwei Protokolle. Das *Sign-In*-Protokoll dient zur Authentifikation des Benutzers und das *Wallet*-Protokoll zur Abwicklung von Online-Einkäufen. Im Folgenden wird kurz auf das *Sign-in*-Protokoll eingegangen. Das *Wallet*-Protokoll wird aufgrund

[11] `http://www.hotmail.com`

der Ähnlichkeit zum *Sign-in*-Protokoll hier nicht weiter beschrieben (weiterführende Informationen dazu finden sich in [62]).

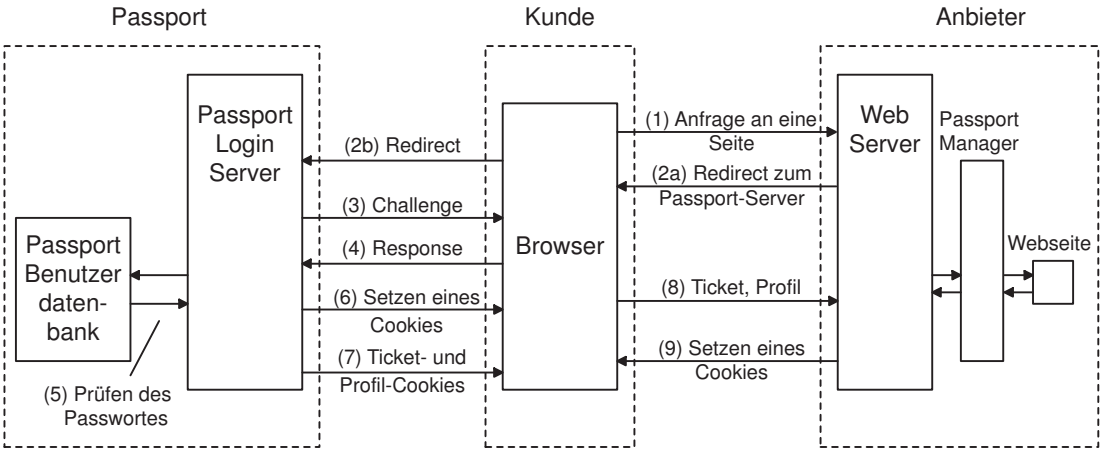

Abbildung 2.22: Single Sign-On mit Passport

Die Anbieter, die den Passport-Dienst nutzen möchten, können zwischen verschiedenen Sicherheitsstufen auswählen. Das *Standard Sign-in* ist für Anbieter gedacht, die keine sicherheitsrelevanten Daten verteilen, jedoch trotzdem einen personalisierten Zugriff auf ihre Daten realisieren möchten. SSL wird beim *Standard Sign-in* nur bei der Übertragung der Benutzerkennung und des Passworts zum Passport-Server genutzt. Die nächste Stufe bildet das *Secure Channel Sign-In*, bei dem die gesamte Übertragung von Daten mittels SSL verschlüsselt ist. Dazu gehört auch die Übertragung des Cookies des Anbieters am Ende des Anmeldevorgangs zum Benutzer. Das *Strong Credential Sign-In* gewährleistet die höchste Sicherheit. Hier muss nach dem Standardanmeldevorgang zusätzlich noch eine vierstellige *Persönliche Identifikationsnummer PIN* eingegeben werden. Wird diese fünfmal fehlerhaft eingegeben, so kann sich ein Benutzer nur noch bei Anbietern anmelden, die das *Standard Sign-In* gewählt haben. Zur Entsperrung muss der Benutzer drei von zehn zuvor vereinbarten Frage/Antwort-Paaren korrekt beantworten. Die PIN ist anbieterabhängig und wird bei der ersten Registrierung eines Benutzers bei dem Anbieter vergeben.[62]

Der Ablauf des Passport-Protokolls ist in Abbildung 2.22 (vgl. [62, 21]) dargestellt. Der Benutzer versucht im Schritt (1) auf die Webseite eines Anbieters zuzugreifen, der die Authentifikation über den Passport-Dienst abwickelt. Der Browser wird daraufhin auf die Webseite des Passport-Servers per *HTTP-Redirect* umgeleitet (2a,2b). Der Passport-Dienst überprüft nun die mitgelieferte ID des Anbieters und fragt den Benutzer nach dessen E-Mail und Passwort (3), falls es sich bei der ID des Anbieters um eine gültige ID eines Passport-Teilnehmers handelt. Die Antworten (*Response*) werden im Protokollschritt (4) per SSL verschlüsselt vom Benutzer zum Passport-Server übertragen. Nach erfolgreicher Authentifikation (5) setzt der Server ein Cookie (*MSPSec*-Cookie) beim Browser des Benutzers (6). Dieses Cookie enthält das verschlüsselte Passwort des Benutzers und damit die Informationen, mit denen der Server im Weiteren den Benutzer automatisch identifizieren kann. Zusätzlich generiert der Server ein Zugriffsticket, den so genannten Passport (u. a. ist hier die PUID enthalten) und setzt (7) diesen wieder in Form eines Cookies (*MSPAuth*-Cookie) im Browser des Benutzers. Der Inhalt dieses Cookies wird mittels 3DES chiffriert. Der zugehörige Schlüssel muss dem Passport-Server sowie allen Anbietern bekannt sein, damit die chiffrierten Daten in den Cookies entschlüsselt werden können. Dadurch

wird die Authentifikation des Benutzers gegenüber dem Anbieter ermöglicht. Das verschlüsselte Ticket und evtl. das Benutzerprofil werden per Queryparameter an die Rückkehr-URL (*Uniform Resource Locator*) angehängt und der Benutzer wird zu dieser URL (die Seite des Anbieters) weitergeleitet. Der Anbieter extrahiert die Daten aus den Parametern, dechiffriert sie und kann den Benutzer anhand der übermittelten PUID in seinem eigenen Datenbestand ermitteln. Nun setzt der Anbieter noch ein Cookie im Browser des Benutzers, um bei nachfolgenden Zugriffen den Kunden zu identifizieren.[21, 62]
Der Benutzer kann sich nach Nutzung der Dienstleistungen durch Aktivieren einer Schaltfläche vom Passport-Dienst abmelden. Der Passport-Server aktiviert dafür bei den Anbietern bei denen sich der Benutzer authentifiziert hat ein Skript, das die jeweiligen Cookies der Anbieter beim Kunden wieder entfernt. Die Cookies des Passport-Dienstes beim Benutzer werden ggf. auch automatisch gelöscht. Die vorgestellten Cookies haben Microsoft-spezifische Namensgebungen. In [63] ist im Abschnitt Cookies eine genaue Auflistung dieser Cookies mit entsprechenden Erklärungen aufgeführt.

Sicherheit Das Passport-Konzept weist teilweise erhebliche Sicherheitsprobleme auf. Größtes Sicherheitsproblem ist die im Mai 2003 durch einen pakistanischen Computer-Spezialisten gefundene Sicherheitslücke. Er hatte entdeckt, dass über eine bestimmte Webseite neue Passwörter für beliebige Passport-Kennungen beantragt werden können und ein Angreifer so in kürzester Zeit kompletten Zugriff auf die Benutzerdaten bekommen konnte.[21]
Passport besitzt noch weitere Sicherheitsprobleme und Schwachstellen [78, 21]:

- Durch Verändern des HTTP-Redirects oder der Veränderung eines *Domain Name System (DNS)*-Eintrags auf dem Benutzer-PC, kann sich ein Angreifer zwischen Kunde und Passport-Server setzen und so alle sicherheitsrelevanten Informationen ausspähen. Eine SSL-Verschlüsselung ist in diesem Fall nicht ausreichend, da diese nur eine Punkt-zu-Punkt-Verschlüsselung bietet.

- Die zentrale Verwaltung der Benutzerprofile macht dies zu einem attraktiven Angriffspunkt.

- Der Passport-Registrierungsdienst fordert von einem Benutzer zwar ein entsprechend sicheres Passwort durch die Kombination von Groß- und Kleinschreibung und die Verwendung von Zahlen und Sonderzeichen, erlaubt aber auch die Verwendung von unsicheren Passwörtern.

- Das Passport-Protokoll realisiert das SSO auf Basis von Cookies. Ist die Verwendung von Cookies im Browser deaktiviert, kann ein Benutzer Passport nicht verwenden.

- Die Verwendung von Cookies ohne die Kombination mit einem ähnlichen Konzept, wie beispielsweise die des Kerberos-Authentifikators, ermöglicht den Missbrauch des Benutzerkontos. Da Passport nicht überprüft, ob ein Benutzer auch rechtmäßiger Besitzer des Cookies ist, kann ein Angreifer es verwenden, um sich als diesen Benutzer auszugeben. Dies ist gerade dann ein Problem, wenn ein Benutzer auf einem öffentlichen Computer den Passport-Dienst nutzt und vergisst, sich vom Passport-Server abzumelden. Durch den Verbleib der Cookies kann das Benutzerkonto missbraucht werden. Auch auf anderen Wegen könnte ein Angreifer in den Besitz des Cookies kommen und so Passport kompromittieren.

- Ab der Stufe *Strong Credential Sign-In* verliert Passport sogar seine ursprüngliche Bestimmung durch die Verwendung von anbieterabhängigen PINs. Ein SSO ist damit nicht mehr möglich, da sich der Benutzer bei jedem Anbieter durch eine PIN extra authentifizieren muss.

2.5.3 Liberty Alliance Project

Das *Liberty Alliance Project*[12], oder kurz *Liberty* ist eine Initiative zur Schaffung eines industrieweiten SSO-Standards. Die Initiative wurde im Jahr 2001 von 33 Firmen (u. a. SUN, Nokia, Cisco, Verisign, ...) gegründet. Das Liberty-Projekt hat im Jahr 2001 die erste Spezifikation 1.0 vorgestellt und im Jahr 2006 eine Folgeversion veröffentlicht.
Die Spezifikation besteht aus den Teilen *Liberty Identity Federation Framework (ID-FF)* [85], *Liberty Identity Web Services Framework (ID-WSF)* [87] und *Liberty Identity Services Interface Specifications (ID-SIS)* [86]. Sie definiert eine Infrastruktur um Nutzer von Internet Anwendungen zu authentifizieren. Im Gegensatz zu Passport werden die Daten der Benutzer dezentral verwaltet. Die unterschiedlichen Service- und Identitäts-Provider der Liberty-Architektur bilden einen so genannten *Trust Circle*.
Die Online-Identität und die dazugehörigen Daten, wie beispielsweise ein Benutzerprofil, eine Zugriffshistorie oder auch die Einkaufsvorlieben, werden in der Liberty-Architektur als *Federated Network Identity* bezeichnet und können von den Benutzern selbst konfiguriert und verwaltet werden. Der Benutzer kann dabei selbst festlegen, welche Informationen über seine Identität von welchen Unternehmen eingesehen werden darf. Zudem legt der Benutzer auch fest, welche Unternehmen im SSO-Prozess berücksichtigt werden. Er definiert also welche Dienstleistungen nach einem SSO direkt ohne weitere Authentifikation von ihm genutzt werden können.[21]
Das Liberty-Konzept greift weitgehend auf Standardprotokolle, die bereits aus dem Web Service-Umfeld bekannt sind, zurück. Dazu gehören beispielsweise HTTP, SSL, SOAP und SAML. Dadurch ist das Liberty-Konzept für einen breiten Einsatz im Internet gewappnet.

2.5.4 Security Assertions Markup Language

Die *Security Assertions Markup Language (SAML)* wurde im Jahr 2002 von der Organisation OASIS als Standard verabschiedet und liegt mittlerweile seit März 2005 in der Version 2 vor. Sie ist aus den Sprachen *Security Services Markup Language (S2ML)* der Firma *Netegrity*[13] und *Authentication Markup Language (AuthML)* entwickelt worden. SAML ist eine auf XML basierende Sprache zur Darstellung sicherheitsrelevanter Informationen, die zwischen unterschiedlichen Instanzen in Form von Zusicherungen (*Assertions*) ausgetauscht werden. SAML kann sowohl zur Realisierung eines SSO-Mechanismus als auch für die Abwicklung von verteilten Transaktionen und zum Aufbau eines Autorisierungsdienstes eingesetzt werden.
SAML ist in vier Hauptbestandteile aufgeteilt, deren Zusammenhang in Abbildung 2.23 (vgl. [90]) dargestellt ist.

- **Assertions**: Die *SAML-Assertions* bilden eine Menge von Zusicherungsanweisungen, die in Form von XML-Schemata in [43] spezifiziert sind. Assertions werden von *SAML-Autoritäten* ausgestellt und verwendet. SAML-Autoritäten, die Assertions ausstellen, werden als *SAML-Asserting Parties* und SAML-Autoritäten, die Assertions empfangen, als *SAML-Relying Parties* bezeichnet.

[12] http://www.projectliberty.org
[13] http://www.netegrity.com

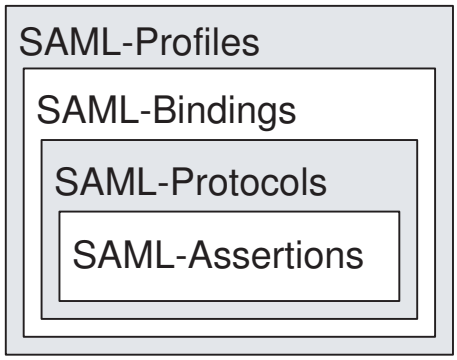

Abbildung 2.23: Bestandteile von SAML

Eine Assertion besteht immer aus einer Versionsnummer (`Version`), die angibt mit welcher Version von SAML die Assertion erstellt wurde. Außerdem einer `AssertionID`, die eine Assertion über eine Identifikationsnummer eindeutig identifiziert und damit Assertions referenzierbar macht. Zudem beinhaltet sie den Zeitpunkt (`IssueInstant`) an dem sie ausgestellt wurde und einen Wert, der angibt von wem sie ausgestellt wurde (`Issuer`). Außerdem kann eine Assertion die Elemente `Subject`, `Conditions`, `Advice` und `ds:Signature` enthalten. `Subject` bindet eine Assertion eindeutig an ein Subjekt. Mittels des `Conditions`-Elements kann die Gültigkeit, in Form von Zeitstempeln und vordefinierten Typen, eingeschränkt werden. Das `Advice`-Element kann als Container für beliebige Daten verwendet werden und über das `ds:Signature`-Element kann eine digitale Signatur, auf Basis von XMLdsig (vgl. Kapitel 2.7.1), eingefügt werden.

Eine Assertion besitzt des Weiteren noch die spezifischen Elemente `AuthnStatement`, `AuthzDecisionStatement`und `AttributeStatement`, die abhängig vom Typ der Assertion verwendet werden. Diese Elemente müssen jeweils mindestens einmal in einer Assertion auftreten. Die Spezifikation definiert drei unterschiedliche Typen von Assertions:

- *Authentication Assertions*: Die Zusicherungsanweisung bestätigt, dass ein Subjekt zu einem bestimmten Zeitpunkt, unter Verwendung einer bestimmten Authentifikationsmethode, authentifiziert wurde.
- *Attribute Assertions*: Die Attributzusicherungsanweisung erlaubt es, weitere Informationen über ein Subjekt zu spezifizieren.
- *Authorization Decision Assertions*: Die Autorisierungsentscheidung definiert, ob ein Objekt auf eine spezifizierte Ressource zugreifen darf oder nicht.

- **Protokolle**: Die Protokolle definieren in [43] anhand von XML-Schemata zunächst wie *SAML-Assertions* angefordert und übermittelt werden. Dazu definieren sie die Interaktion zwischen einem so genannten *SAML-Requester* (Anfrager) und einem *SAML-Responder* (Antworter) sowie den Aufbau eines *Request* und eines *Response*. Der SAML-Protokollteil umfasst die Spezifikation der folgenden Protokolle:
  - *Assertion Query and Request Protocol*
  - *Authentication Request Protocol*

  - *Artifact Resolution Protocol*
  - *Name Identifier Management Protocol*
  - *Single Logout Protocol*
  - *Name Identifier Mapping Protocol*

- **Bindings**: Die SAML-Bindings [44] beschreiben generische Protokolle, die an unterschiedliche Standardprotokolle gebunden werden können. Die *SAML-Bindings* legen fest, wie die spezifizierten Protokolle über Standardübertragungsprotokolle abgebildet werden. Bekanntestes Binding ist das *SOAP-über-HTTP-Binding*. Es legt fest, wie SOAP in Kombination mit HTTP eingesetzt werden kann, um eine Anfrage an eine so genannte *SOAP-Authority* zu stellen und eine Antwort zu erhalten. Neben diesem Binding existieren in der Spezifikation noch weitere Bindings:

  - *Reverse SOAP (PAOS) Binding* [88]
  - *HTTP Redirect Binding*
  - *HTTP POST Binding*
  - *HTTP Artifact Binding*
  - *SAML URI Binding*

- **Profile**: Die SAML-Profile [45] spezifizieren wie SAML-Assertions, SAML-Protokolle und SAML-Bindings zusammen für einen bestimmten Anwendungsfall benutzt werden können. Das bekanntestes Profil ist das *Web Browser SSO Profile* (vgl. Abschnitt 2.5.4.1). Die spezifizierten Profile unterteilen sich wie folgt:

  - *SSO Profiles of SAML*
  - *Artifact Resolution Profile*
  - *Assertion Query/Request Profile*
  - *Name Identifier Mapping Profile*
  - *SAML Attribute Profiles*

2.5.4.1 SSO mit SAML

Um einen SSO-Mechanismus mit SAML zu ermöglichen, kann das *Web Browser SSO Profile* verwendet werden.
Abbildung 2.24 stellt exemplarisch dar wie dies realisiert wird. Die einzelnen Schritte werden durch das Profil beschrieben und in jedem Schritt können jeweils mehrere Nachrichten, abhängig vom genutzten Binding, ausgetauscht werden.

1. Ein noch nicht authentifizierter Benutzer schickt im Schritt (1) eine HTTP-Anfrage an einen Service-Provider, um auf eine geschützte Ressource zuzugreifen.

2. Der *Service-Provider* stellt in Schritt (2) fest, dass der Benutzer noch nicht authentifiziert ist und determiniert den Ort des zuständigen *Identity-Provider*. Dies ist implementierungsabhängig und kann mit dem in der Spezifikation definierten *SAML Identity Provider Discovery Profile* [45] durchgeführt werden.

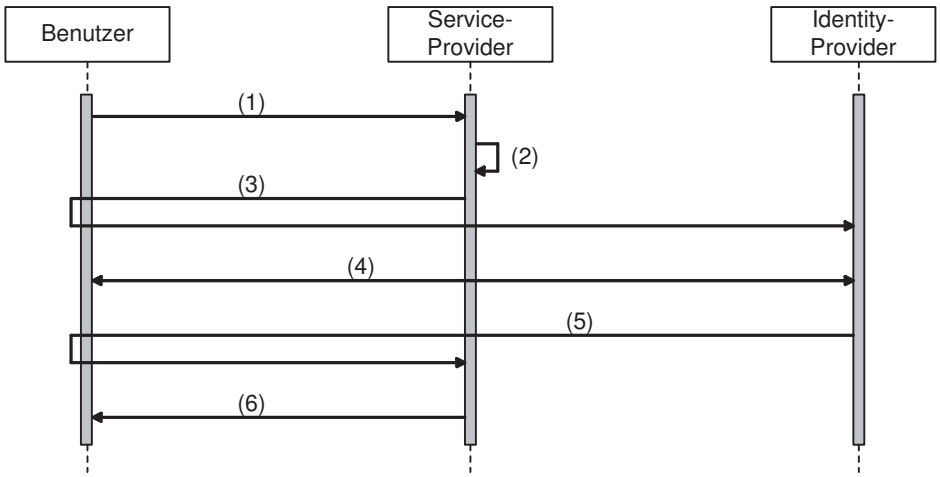

Abbildung 2.24: SAML-SSO: Web Browser SSO Profile

3. In Schritt (3) stellt der Service-Provider eine `AuthnRequest`-Nachricht aus, die über den Benutzer an den Identity-Provider versendet wird. Dies kann mit Hilfe der Bindings *HTTP Redirect*, *HTTP POST*, oder *HTTP Artifact* erfolgen.

4. Die Authentifikation des Benutzers durch den Identity-Provider wird im Schritt (4) vorgenommen. Wie dies geschieht wird durch das Profil nicht weiter definiert.

5. Der Identity-Provider stellt in Schritt (5) eine `Response`-Nachricht aus, die dann über den Benutzer an den Service-Provider zugestellt wird. Hierzu kann wieder das Binding *HTTP POST* oder *HTTP Artifact* verwendet werden. Die Nachricht enthält bei erfolgreicher Authentifikation mindestens eine *Authentication Assertion*, bzw. bei einem Fehler eine entsprechende Fehlermeldung. Das *HTTP Redirect* Binding kann hier nicht eingesetzt werden, da die Nachrichtenlänge und die sich dadurch ergebende URL die zulässige Länge einer URL überschreitet.

6. Aufgrund der erhaltenen Assertion bzw. Fehlermeldung kann der Service-Provider nun den Zugriff auf die gewünschte Ressource erlauben oder verweigern.

2.6 Kryptografische Grundlagen

Die Kryptografie ist die Lehre von den Methoden zur Verschlüsselung und Entschlüsselung von Nachrichten zum Zweck der Geheimhaltung von Informationen gegenüber Dritten. Als Kryptoanalyse wird die Wissenschaft der Überwindung kryptografischer Verfahren bezeichnet. Sie wird zur Entschlüsselung von Nachrichten eingesetzt, wenn kein Zugriff auf die verwendeten Schlüssel möglich ist. Kryptografie und Kryptoanalyse können unter dem Überbegriff Kryptologie zusammengefasst werden. Neben diesen Teilbereichen gewinnt die Steganografie im Umfeld der Kryptologie zunehmend an Bedeutung. Die Steganografie versucht bereits die Existenz einer Nachricht zu verbergen und nicht nur den Informationsgehalt einer Nachricht zu verschleiern. Die Steganografie erlaubt es, auf aufwendige kryptografische Mechanismen zu verzichten.
Die Kryptoanalyse und die Steganografie haben für das Umfeld dieser Arbeit keine besondere Bedeutung und werden daher nicht detailliert betrachtet. Stattdessen werden nun die wichtigsten

kryptografischen Grundlagen, die für die weitere Betrachtung notwendig sind, vorgestellt. Dazu werden zunächst die grundlegenden Verschlüsselungsverfahren beschrieben. Danach erfolgt eine Beschreibung des Konzepts der kryptografischen Hash-Funktion und der elektronischen Signatur. Abschließend wird noch ein Überblick über digitale Zertifikate gegeben.
Die Kryptografie ermöglicht es, einige der wichtigsten Schutzziele (vgl. Kapitel 2.1.2) zu gewährleisten. Durch den Einsatz von Verschlüsselungsmechanismen kann die Vertraulichkeit von Nachrichten sichergestellt werden. Außerdem werden sie eingesetzt, um im Bereich der Authentifikation und Identifikation eine sichere Abspeicherung von Passwörtern bzw. Benutzerinformationen zu gewährleisten. Die Integrität von Daten kann durch die Anwendung von digitalen Fingerabdrücken erreicht werden, die durch den Einsatz von kryptografischen Hash-Funktionen erzeugt werden. Um das Schutzziel der Verbindlichkeit zu erreichen, können mit Hilfe der digitalen Signatur Aktionen eines Systems oder eines Benutzers verbindlich zugeordnet werden.

2.6.1 Verschlüsselung und Entschlüsselung

Für die Verschlüsselung (*Chiffrierung*) bzw. Entschlüsselung (*Dechiffrierung*) von Daten werden so genannte kryptografische Systeme (im Folgenden als Kryptosystem bezeichnet) eingesetzt. Diese legen einerseits fest, wie der Klartext einer Nachricht in einen Geheimtext transformiert (verschlüsselt) werden kann. Zudem wird durch ein Kryptosystem auch festgelegt, wie der Geheimtext einer Nachricht wieder in Klartext zurück transformiert (entschlüsselt) werden kann. Dadurch erlauben Kryptosysteme Informationen vor unbefugtem Zugriff zu schützen. Kryptosysteme verwenden für die Entschlüsselung und Verschlüsselung von Nachrichten einen Schlüssel. Mit der folgenden Definition (vgl. [21]) eines Kryptosystems kann jedes bekannte Verschlüsselungssystem beschrieben werden.

Definition 2.11 *Kryptografisches System (Kryptosystem):*
Gegeben seien zwei endliche Zeichenvorräte (Alphabete) A_1 und A_2. Ein Kryptosystem ist gegeben durch ein Tupel

$$\mathcal{KS} = (\mathcal{M}, \mathcal{C}, EK, DK, E, D) \textit{ mit}$$

1. *der nicht leeren endlichen Menge von Klartexten $\mathcal{M} \subseteq A_1^*$ wobei A_1^* die Menge aller Worte über dem Alphabet A_1 beschreibt,*
2. *der nicht leeren endlichen Menge von Geheim-, Krypto- bzw. Chiffretexten $\mathcal{C} \subseteq A_2^*$,*
3. *der nicht leeren Menge von Verschlüsselungsschlüsseln EK,*
4. *der nicht leeren Menge von Entschlüsselungsschlüsseln DK*
   sowie einer Bijektion $f : EK \longrightarrow DK$. Die Bijektion assoziiert zu einem Verschlüsselungsschlüssel $K_E \in EK$ einen dazu passenden Entschlüsselungsschlüssel $K_D \in DK$, d.h. $f(K_E) = K_D$,
5. *dem linkstotalen und injektivem Verschlüsselungsverfahren*

$$E : \mathcal{M} \times EK \longrightarrow \mathcal{C} \textit{ und}$$

6. *dem Entschlüsselungsverfahren*

$$D : \mathcal{C} \times DK \longrightarrow \mathcal{M},$$

mit der Eigenschaft, dass für zwei Schlüssel $K_E \in EK, K_D \in DK$ mit $f(K_E) = K_D$ gilt:

$$\forall M \in \mathcal{M} : D(E(M, K_E), K_D) = M.$$

Die Definition 2.11 definiert damit Klartext und Chiffretext als Worte über einem endlichen Zeichenvorrat, die aus den Alphabeten A_1 und A_2 gebildet werden können. Außerdem zeigt die Definition, dass ein zu einem Verschlüsselungsschlüssel K_E passender Entschlüsselungsschlüssel $K_D = f(K_E)$ zum entschlüsseln einer in Chiffretext vorliegenden Nachricht verwendet werden kann. Schlüssel für die Entschlüsselung und Verschlüsselung werden dabei aus einem zulässigen Schlüsselraum gebildet.
Die Sicherheit eines Kryptosystems wird maßgeblich durch die verwendeten Schlüssel beeinflusst. Ihre Geheimhaltung ist zwingend, um die Vertraulichkeit einer Nachricht zu gewährleisten. Kryptosysteme können auch durch die Geheimhaltung der eingesetzten Mechanismen ihre Sicherheit und damit das Maß an Vertrauen in das System erlangen. Jedoch werden Systeme, die ihre Mechanismen nicht offen darlegen, von der Öffentlichkeit nicht als sehr vertrauenswürdig angesehen und setzen sich daher in der Praxis meist nicht durch. Basiert die Sicherheit eines Kryptosystems außerdem nur darauf, dass ein Angreifer das Verschlüsselungsverfahren nicht kennt, so ist das Verfahren nach dem Bekanntwerden nicht mehr sicher einzusetzen. Deshalb sollte die Sicherheit eines kryptografischen Verfahrens allein von den verwendeten Schlüsseln abhängig sein. Dies wird auch als das Kerckhoffs-Prinzip bezeichnet. Ist das Verfahren bekannt, so muss daher bei der Auswahl von geeigneten Schlüsselräumen zwingend auf eine ausreichende Größe geachtet werden. Dadurch sollen Angriffe durch einfaches Ausprobieren erschwert werden.
Einfache Kryptosysteme sind bereits aus der Geschichte bekannt. Die Caesar-Verschlüsselung, benannt nach dem römischen Kaiser J. Caesar, verwendet einen einfachen Mechanismus um eine Nachricht zu verschlüsseln. Dabei werden die Buchstaben einer Nachricht durch verschieben um eine feste Zeichenzahl (hier um 3) in einen Geheimtext übersetzt. So würde durch die Anwendung der Caesar-Verschlüsselung der Klartext „`IT Sicherheit`“ durch den Geheimtext „`LW Vlfkhukhlw`“ ersetzen werden. Für die Verschlüsselung gilt

$$E(M) = (M + K_E) \bmod 26 = C, \text{ mit } M \in A = \{a, ..., z\}, K_E = 3$$

und für die Entschlüsselung gilt

$$D(C) = (C + K_D) \bmod 26 = M, \text{ mit } C \in A = \{a, ..., z\}, K_D = f(K_E) = -K_E.$$

Verfahren, die auf solchen einfachen Schiebemechanismen basieren, werden auch als monoalphabetische Substitutionsverfahren bzw. „Shift cipher“ bezeichnet.
Zu den polyalphabetischen Substitutionsverfahren gehört das *One-Time-Pad*-Verfahren (Einmalverschlüsselung). Bei diesem Verfahren wird jeweils einem Buchstaben oder Zeichen des Klartextes, ein durch einen Schlüssel definierter anderer Buchstabe bzw. Zeichen zugeordnet. Hierbei muss der Schlüssel mindestens die gleiche Länge wie der Klartext haben und darf nur einmal verwendet werden.
Beide Verfahren besitzen jedoch einige Nachteile. So sind Systeme, die auf monoalphabetischer Substitution basieren, relativ einfach überwindbar. Durch einfaches Analysieren der Häufigkeitsverteilung kann relativ schnell auf den Klartext geschlossen werden. Polyalphabetische Substitutionsverfahren, wie beispielsweise die Vigenère-Verschlüsselung, sind mathematisch

zwar als sicher bewiesen, jedoch kann durch Verletzung einiger Voraussetzungen, wie das Ausspähen des Schlüssels, Verwendung von Schlüsseln mit unzureichender Länge oder auch Mehrfachverwendung, auf den Klartext geschlossen werden. Ein weiterer Nachteil von Kryptosystemen, die auf dem polyalphabetischen Substitutionsverfahren basieren, ist die Größe des zu verwendeten Schlüssels. Gerade bei sehr großen Nachrichten ist die Verwendung eines mindestens gleich großen Schlüssels sehr ineffizient.
Die vorgestellten Systeme basieren auf klassischen, symmetrischen Verfahren deren Mechanismen im Folgenden vorgestellt werden.

Symmetrische Kryptosysteme Die symmetrischen Kryptosysteme zeichnen sich dadurch aus, dass sie für die Chiffrierung und Dechiffrierung die gleichen Schlüssel (*secret keys* oder *shared keys*) verwenden. Benutzen zwei Kommunikationspartner ein symmetrisches Verschlüsselungssystem so müssen beide Partner einen gemeinsamen Schlüssel vereinbaren. Dies bedeutet, dass der Kommunikationspartner, der eine chiffrierte Nachricht erhält und diese dechiffrieren möchte, den Schlüssel benötigt, der auch zum chiffrieren verwendet wurde. Dazu ist es notwendig, dass der Schlüssel auf einem sicheren Weg zum Empfänger der verschlüsselten Nachricht gelangt. Die Sicherheit eines symmetrischen Systems ist also stark davon abhängig wie geheime Schlüssel aufbewahrt, verwaltet und ausgetauscht werden.
Symmetrische Verfahren verwenden für die Ver- und Entschlüsselung Algorithmen, die auf sehr einfachen Operationen basieren. Dies erlaubt es, diese Algorithmen sehr effizient in Software und auch in Hardware zu implementieren. Sie eigenen sich daher sehr gut, um große Datenmengen schnell und effizient zu ver- bzw. entschlüsseln.
In Abbildung 2.25 ist die Chiffrierung und Dechiffrierung von Nachrichten unter Anwendung eines symmetrischen Kryptosystems dargestellt.

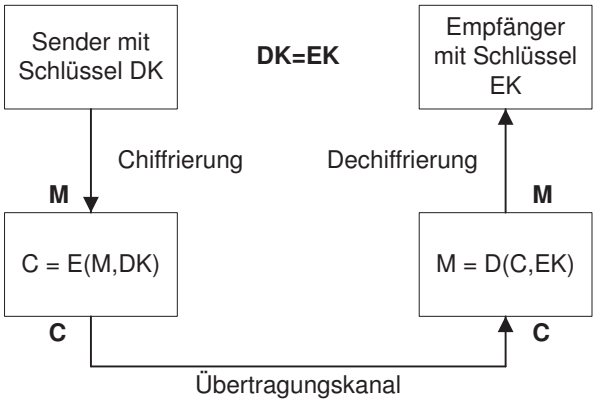

Abbildung 2.25: Symmetrische Verschlüsselung

Ein großer Nachteil dieser Systeme ist die große Anzahl von Schlüsseln, die benötigt werden, wenn viele Kommunikationspartner verschlüsselte Nachrichten austauschen möchten. Die Anzahl der Schlüssel steigt quadratisch mit der Anzahl der Teilnehmer. Genauer ausgedrückt werden für k Teilnehmer $n = \frac{k(k-1)}{2}$ Schlüssel benötigt. Dies bedeutet, dass bei nur 20 Teilnehmern schon 190 Schlüssel benötigt werden. Wollen alle Einwohner einer mittleren Kleinstadt mit 50.000 Bewohnern verschlüsselte Nachrichten austauschen, so werden bereits 1.249.975.000 Schlüssel benötigt. Es wird deutlich, dass die Schlüsselverwaltung bei einem symmetrischen Kryptosystem eine komplizierte Angelegenheit werden kann.

Zu den wichtigsten und bekanntesten symmetrischen Verschlüsselungsverfahren zählen der *Data Encryption Standard (DES)* und dessen Variante *Triple Data Encryption Standard (3DES)*, der *Advanced Encryption Standard (AES)* sowie der *International Data Encryption Algorithm (IDEA)*. AES ist auch unter dem Name Rijndael bekannt, was auf die Nachnamen seiner Entwickler Vincent Rijmen und Joan Daemen zurückzuführen ist. In Tabelle 2.7 sind die wichtigsten Eigenschaften dieser Verfahren aufgelistet.

Algorithmus	Blocklänge	Schlüssellänge	Schlüsselraumgröße
DES	64 Bit	64 Bit (8 Paritätsbits)	2^{56}
3DES	64 Bit	168 Bit	2^{108}
AES (Rijndael)	128, 192, 256 Bit	128, 192, 256 Bit	2^n $(n = 128, 192, 256)$
IDEA	64 Bit	128 Bit	2^{128}

Tabelle 2.7: Symmetrische Verschlüsselungsalgorithmen

Neben den klassischen symmetrischen Kryptosystemen existieren die modernen, asymmetrische Kryptosysteme.

Asymmetrische Kryptosysteme Asymmetrische Kryptosysteme haben ihren Ursprung in der Mitte der 70er Jahre. Durch Ralph Merkle sowie Diffie und Hellman wurde die Idee für eine neue Klasse von Kryptosystemen entwickelt. Die Neuigkeit der asymmetrischen Kryptosysteme besteht darin, dass für die Ver- und Entschlüsselung jeweils unterschiedliche Schlüssel verwendet werden können. Dazu erhält jeder Kommunikationspartner ein Schlüsselpaar bestehend aus einem persönlichen, geheimen Schlüssel (*private key*) und einem öffentlichen Schlüssel (*public key*). Obwohl beide Schlüssel eine gemeinsame mathematische Basis besitzen, ist es nicht möglich, von einem Schlüssel auf den anderen zu schließen.

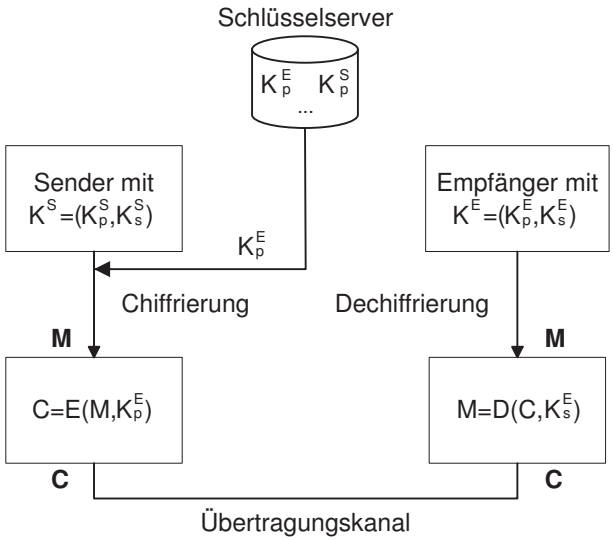

Abbildung 2.26: Asymmetrische Verschlüsselung

Bei asymmetrischen Kryptosystemen wird jeweils das Schlüsselpaar K^E des Empfängers für die Chiffrierung und Dechiffrierung der Nachricht verwendet. Dazu benutzt der Sender zur

Chiffrierung den öffentlichen Schlüssel K_p^E des Empfängers. Nach dem der Empfänger die verschlüsselte Nachricht erhalten hat, kann er die Nachricht mit seinem privaten Schlüssel K_s^E dechiffrieren. Dieser Sachverhalt wird in Abbildung 2.26 veranschaulicht.

Asymmetrische Kryptosysteme verringern die benötigte Schlüsselanzahl um ein Vielfaches. Durch die Verwendung von Schlüsselpaaren, bestehend aus öffentlichem und privatem Schlüssel, wird pro Teilnehmer nur ein Schlüsselpaar und damit nur 2 Schlüssel benötigt. Dies vereinfacht das Schlüsselmanagement erheblich. Jedoch muss auch sichergestellt werden, dass der Sender einer Nachricht den öffentlichen Schlüssel des Empfängers einer Nachricht auf einfache Weise beziehen kann. Dies wird beispielsweise durch die Bereitstellung eines Schlüssel-Servers ermöglicht, der alle öffentlichen Schlüssel der Teilnehmer speichert und für die Chiffrierung zur Verfügung stellt. Damit der Sender einer Nachricht sicher sein kann, dass ein verwendeter öffentlicher Schlüssel auch wirklich dem gewünschten Empfänger der Nachricht gehört, muss es Möglichkeiten geben die dies garantieren. Zur Sicherstellung und Kontrolle der Schlüsselauthentizität existiert das Konzept der digitalen Zertifikate, das in Kapitel 2.6.4 vorgestellt wird.

Wie bereits erwähnt besitzen private und öffentliche Schlüssel eine gemeinsame Basis. Dadurch ist es notwendig, dass die verwendeten Schlüssel erheblich länger als bei der symmetrischen Verschlüsselung sind. Werden bei der symmetrischen Verschlüsselung Schlüssellängen zwischen 64 und 256 Bit verwendet, so muss, um die gleiche Sicherheit zu gewährleisten, bei der asymmetrischen Verschlüsselung mit erheblich längeren Schlüssellängen gearbeitet werden. So muss für ein asymmetriches Verfahren mit einer Schlüssellänge von 2.304 Bit (768 Bit, 512 Bit, 384 Bit) gearbeitet werden, um die gleiche Sicherheit zu gewährleisten wie mit einem symmetrischen Verfahren mit einer Schlüsselänge von 128 Bit (80 Bit, 64 Bit, 56 Bit) (vgl. [95]).

Im Jahr 1978 wurde der erste Algorithmus, der die Idee eines asymmetrischen Kryptosystems umsetzt, implementiert. Der RSA-Algorithmus wurde von Ronald L. Rivest, Adi Shamir und Leonard M. Adleman implementiert und ist bis heute der de-facto Standard. Dies ist auf seine große Verbreitung und Akzeptanz zurückzuführen. Neben RSA existieren noch weitere asymmetrische Kryptosysteme wie beispielsweise das Rabin- oder das Elgamal-Kryptosystem.

Ein Nachteil der asymmetrischen Verfahren ergibt sich aus dem Algorithmus und dessen Operationen. Das RSA-Verfahren ist in Abhängigkeit davon, ob eine Hardware- bzw. Softwarelösung vorliegt um einen Faktor von 100 bzw. 1000 langsamer als beispielsweise das symmetrische DES-Verfahren. Dadurch ist der Rechenaufwand gerade bei großen Nachrichten für die Chiffrierung und Dechiffrierung sehr hoch. In der Praxis wird deshalb eine Kombination beider Verfahren bevorzugt. Diese Systeme werden als hybride Verschlüsselungsverfahren bezeichnet.

Hybride Verfahren Bei der hybriden Verschlüsselung werden gezielt die Vorteile der symmetrischen und asymmetrischen Verschlüsselungsverfahren ausgenutzt. Hierbei wird die Geschwindigkeit der symmetrischen Verfahren mit der Sicherheit der asymmetrischen Verfahren kombiniert. In der Praxis wird die Nachricht durch den Einsatz eines symmetrischen Kryptosystems schnell und effizient verschlüsselt. Das Problem des Schlüsselaustauschs wird durch den Einsatz eines zusätzlichen asymmetrischen Kryptosystems beseitigt. Dazu wird einfach der geheime Schlüssel, der bei der symmetrischen Verschlüsselung verwendet wurde, mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Da die Größe des Schlüssels nur ein Bruchteil der Größe der Nachricht darstellt, fällt hier die schlechte Leistungsfähigkeit des asymmetrischen Verfahrens nicht ins Gewicht. Der Empfänger der Nachricht kann nun mit seinem privaten Schlüssel des asymmetrischen Kryptosystems den geheimen Schlüssel des symmetrischen Verfahrens dechiffrieren und mit diesem Schlüssel dann die Nachricht entschlüsseln. Durch dieses Verfahren wird auch der Nachteil der Schlüsselverwaltung und Schlüsselaufbewahrung des

symmetrischen Verfahrens beseitigt, da für jede Sitzung ein neuer symmetrischer Schlüssel verwendet werden kann.
Für den Fall, dass eine Nachricht an mehrere Personen verschlüsselt gesendet werden soll, kann auch hier die hybride Verschlüsselung sehr effizient eingesetzt werden. Hierbei wird der eingesetzte symmetrische Schlüssel jeweils einmal mit den öffentlichen Schlüsseln der Empfänger verschlüsselt und der Nachricht beigefügt. Dadurch wird auf relativ einfache und schnelle Weise die Nachricht für mehrere Empfänger verschlüsselt, ohne dass die Nachricht für jeden Empfänger einzeln verschlüsselt werden muss.
Die Anwendung der hybriden Verschlüsselung findet in der Praxis eine weite Verbreitung. Neben dem *Secure Socket Layer (SSL)* Protokoll (vgl. Kapitel 2.8.1), das unter anderem in Browsern für die sichere Verbindung beim Online-Banking eingesetzt wird, werden hybride Verfahren, beispielsweise auch in dem Softwarepaket *Pretty Good Privacy (PGP)* [35] sowie dem IPSec-Netzwerkprotokoll [26], eingesetzt.

2.6.2 Hash-Funktionen

Um das Schutzziel der Integrität zu gewährleisten, wird eine Möglichkeit benötigt unautorisierte Modifikationen, die auf dem Transportweg einer Nachricht vorgenommen wurden, eindeutig zu erkennen. Dazu werden so genannte Hash-Funktionen eingesetzt. Eine Hash-Funktion berechnet von einer Nachricht einen digitalen Fingerabdruck. Der Empfänger einer Nachricht kann daraufhin anhand des mitgeschickten Fingerabdrucks überprüfen, ob eine Modifikation der Nachricht vorgenommen wurde.
Hash-Funktionen basieren auf so genannten Einwegfunktionen. Diese bilden eine Eingabe auf genau eine Ausgabe ab und erlauben es nicht, von der Ausgabe wieder auf die Eingabe zu schließen. Durch eine kryptografische Hash-Funktion wird von einer Nachricht beliebiger Länge meist durch Kompression eine Prüfsumme fester Länger berechnet. In die Berechnung der Prüfsumme geht dabei jedes einzelne Bit der Nachricht ein, so dass die Veränderung nur eines einzigen Bits bereits erkannt werden kann. Eine kryptografische Hash-Funktion muss außerdem kollisionsfrei arbeiten. Das heißt es darf nicht möglich sein, zwei unterschiedliche Eingaben zu finden, die denselben Hash-Wert besitzen.
Zu den bekanntesten Verfahren zur Berechnung von Hash-Werten gehören die *Message Digest (MD)* Verfahren sowie der *Secure Hash Algorithm* SHA-1. Der MD5-Algorithmus [91] wurde 1991 von R. Rivest eingeführt und liefert einen 128-Bit Hash-Wert. Der SHA-1 [19] wurde im Jahr 1993 veröffentlicht erzeugt 160-Bit Hash-Werte.
Bei beiden Verfahren kann jedoch ein Angreifer durch Veränderung der Nachricht und anschließender Neuberechnung des Hash-Werts die Integrität der Daten verletzen, ohne dass dies erkannt werden kann. In der Praxis existiert deshalb das Konzept der *Message Authentication Codes (MAC)*. Bei diesen Verfahren fließt neben der eigentlichen Nachricht zusätzlich ein geheimer Schlüssel in die Berechnung des Hash-Werts ein. Ein Angreifer benötigt damit zusätzliche Kenntnis über den verwendeten geheimen Schlüssel. Da nur die Kommunikationspartner im Besitz des verwendeten geheimen Schlüssels sind, erlaubt das MAC Konzept, neben der Integrität auch die Authentizität des Urhebers der Daten festzustellen.
MD5 und SHA-1 werden mittlerweile als nicht mehr 100 prozentig sicher angesehen. Dies ist zum einen auf die nachweislich nicht vorhandene Kollisionsfreiheit des MD5 Algorithmus und zum anderen auf die Anfälligkeit beider Algorithmen auf *Brute Force*-Attacken zurückzuführen. Das *Keyed-Hashing for Message Authentication (HMAC)*-Verfahren [59] stellt eine Weiterentwicklung dieser Verfahren dar und behebt dabei die Schwächen der MD5 und SHA-1 Algorithmen. Im Gegensatz zum MAC-Verfahren, bei dem die Nachricht mit dem geheimen Schlüssel

verkettet und dann der Hash-Wert gebildet wird, wird beim HMAC-Verfahren der Schlüssel dazu verwendet, die Berechnung des Hash-Werts direkt zu beeinflussen. Dazu wird der Initialwert, mit dem die Anfangswerte der Kompressionsfunktion festgelegt werden, beeinflusst. Daraus ergibt sich, dass Angriffe mittels Brute-Force Attacken sehr viel schwieriger sind. Mit dem aktuellen Stand der Technik ist so eine Attacke nicht in einer zufriedenstellenden Zeit zu bewältigen. Die MD5 und SHA-1 Algorithmen können daher weiterhin in Kombination mit dem HMAC-Verfahren verwendet werden.
Wenn ein Angreifer in den Besitz des geheimen Schlüssels, der bei den MAC- bzw. HMAC-Verfahren verwendet wird, gelangt, kann die Authentizität des Urhebers einer Nachricht nicht mehr gewährleistet werden. Um dies zu vermeiden werden elektronische Signaturen eingesetzt.

2.6.3 Elektronische Signaturen

Als elektronische bzw. digitale Signatur wird ein Mechanismus bezeichnet, der es erlaubt, digitale Dokumente digital zu unterschreiben. Damit eine sichere, digitale Signatur den gleichen Zweck wie eine handschriftliche Unterschrift erfüllt, müssen einige Anforderungen vorausgesetzt werden. Die Signatur muss genauso wie eine Unterschrift die Identität des Unterzeichners zweifelsfrei nachweisen. Außerdem darf sie nicht wiederverwendbar und nur mit dem mit ihr signierten Dokument gültig sein. Es muss zweifelsfrei nachweisbar sein, dass das signierte Dokument dem ursprünglichen Dokument entspricht. Das heißt jede Veränderung des signierten Dokuments muss die Signatur ungültig machen. Letztlich muss eine Signatur verbindlich sein. Ein nachträgliches Abstreiten darf nicht möglich sein.
Prinzipiell lässt sich eine digitale Signatur durch zwei Verfahren realisieren. Durch Anwendung eines symmetrischen Verschlüsselungsverfahrens, kann, in Verbindung mit einer dritten vertrauenswürdigen Instanz, ein signiertes Dokument zwischen zwei Kommunikationspartnern ausgetauscht werden (vgl. [21]). Diese Möglichkeit hat sich jedoch in der Praxis nicht durchgesetzt, da durch den Einsatz von symmetrischen Kryptosystemen auch deren Nachteile zum Tragen kommen. Des Weiteren sind diese Verfahren nur sehr zeitaufwendig einzusetzen und die Vertrauenswürdigkeit der dritten Instanz ist besonders zu schützen.
Durch den Einsatz eines asymmetrischen Verschlüsselungsverfahrens lassen sich die Anforderungen an eine digitale Signatur relativ einfach umsetzen. Dazu muss das eingesetzte asymmetrische Kryptosystem jedoch die Möglichkeit bieten, die öffentlichen und privaten Schlüssel in umgekehrter Reihenfolge zu verwenden. Dies bedeutet, dass es möglich sein muss, ein mit einem privaten Schlüssel chiffriertes Dokument mit dem zugehörigen öffentlichen Schlüssel wieder zu entschlüsseln.
Mit einer digitalen Signatur können die Schutzziele Integrität der Daten, Authentizität des Urhebers sowie die Verbindlichkeit des Nachrichtenursprungs, erfüllt werden:

1. Zunächst wird von einem zu versendeten Dokument ein Hash-Wert gebildet. Dies kann mit den im vorherigen Kapitel beschriebenen Verfahren erfolgen. Prinzipiell wären die genannten Anforderungen an eine digitale Signatur auch ohne die Bildung eines Hash-Werts erfüllbar, jedoch wird aus Effizienzgründen auf die Verwendung von Hash-Funktionen nicht verzichtet.

2. Danach wird der Hash-Wert des Dokuments mit dem privaten Schlüssel des Senders, also dem Unterzeichner, verschlüsselt. Die daraus entstehende Datenmenge wird als Signatur bezeichnet. Zusammen mit dem Dokument wird die Signatur nun an den Empfänger versendet.

3. Nach dem Erhalt des Dokuments versucht nun der Empfänger mit dem öffentlichen Schlüssel die Signatur zu entschlüsseln. Gelingt dies so ist zweifelsfrei nachgewiesen, dass das Dokument tatsächlich von diesem Absender stammt, da ja nur dieser den passenden privaten Schlüssel besitzen kann. Somit ist gleichzeitig die Authentizität des Urhebers sowie der Nachrichtenursprung verbindlich festgestellt.

4. Berechnet der Empfänger seinerseits den Hash-Wert des Dokuments und vergleicht diesen mit dem aus der Signatur entschlüsselten Wert, kann er somit auch die Integrität des Dokuments feststellen. Wäre das Dokument auf dem Transportweg verändert worden, so würde der berechnete Hash-Wert nicht mit dem entschlüsselten Hash-Wert übereinstimmen.

Werden signierte Dokumente unverschlüsselt versendet, so kann ein Angreifer die Signatur entfernen, das Dokument evtl. verändern, eine eigene Signatur erstellen und die so veränderten Daten an den ursprünglichen Empfänger senden. Daher ist es sinnvoll, signierte Dokumente zusätzlich zu verschlüsseln. Außerdem kann der Empfänger eines Dokuments die Identität des Unterzeichners nur dann zweifelsfrei feststellen, wenn er der Tatsache vertraut, dass der öffentliche Schlüssel, der zur Überprüfung einer Signatur verwendet wird, wirklich dem vorgegebenen Unterzeichner gehört. Damit dies sichergestellt werden kann, existiert das Konzept der digitalen Zertifikate.

2.6.4 Digitale Zertifikate

Mit einem digitalen Zertifikat wird die Zuordnung eines öffentlichen Signierschlüssels zu einer Person von einer vertrauenswürdigen Instanz rechtsverbindlich bescheinigt. Dadurch kann die Vertrauenswürdigkeit und Verbindlichkeit einer digitalen Signatur uneingeschränkt erfüllt werden.
Zertifikate werden von so genannten Zertifizierungsstellen (*Certification Authority (CA), Trust Center*) ausgestellt und bei der Ausstellung mit dem privaten Schlüssel der Zertifizierungsstelle signiert. Neben der Ausstellung von Zertifikaten sind diese Stellen auch für deren Überprüfung zuständig. Zertifizierungsstellen generieren des Weiteren Schlüsselpaare für die Teilnehmer und stellen ein Verzeichnis für den Abruf von öffentlichen Schlüsseln zur Verfügung.
Eine Zertifizierungsstelle kann durch unterschiedliche Instanzen repräsentiert werden. Einerseits können sie von speziellen Unternehmen ,wie beispielsweise VeriSign[14], GlobalSign[15] oder Thawte[16], wahrgenommen werden, andererseits können aber auch Institutionen innerhalb eines Unternehmens oder öffentliche Organisationen und Regierungsstellen, wie die Bundesnetzagentur, als Zertifizierungsstelle dienen.
Die Erzeugung und Verwaltung von Zertifikaten wird in der Praxis von einer speziellen Infrastruktur übernommen. Diese wird als *Public Key-Infrastruktur (PKI)* bezeichnet und besteht aus mehreren Zertifizierungs- (CA) und Registrierungsstellen (*Registration Authority (RA)*).
In der Praxis werden heute meist Zertifikate basierend auf dem X.509 Standard [29] eingesetzt. Ein X.509 Zertifikat enthält eine Versionsnummer, einen eindeutigen Namen der den Signierschlüsselinhaber identifiziert, eine Seriennummer der zur eindeutigen Identifizierung des Zertifikats dient, den öffentlichen Signierschlüssel des Inhabers, die verwendeten Signatur-Algorithmen, den Gültigkeitszeitraum sowie den Namen der ausstellenden Zertifizierungsstelle.

[14] http://www.verisign.com
[15] https://globalsign.wis.de
[16] http://www.thawte.com

Zudem kann ein Zertifikat auch spezielle Erweiterungen enthalten. Diese Erweiterungen erlauben es beispielsweise festzulegen, für welchen Zweck (Verschlüsselung, Signierung, Zertifizierung) der enthaltene Schlüssel verwendet werden darf.

2.7 XML Sicherheit

Die Sicherheit von Web Service-Anwendungen hängt maßgeblich von der Sicherheit der zugrunde liegenden Technologie XML ab. Im Folgenden wird daher gezeigt, wie die im vorherigen Kapitel vorgestellten Schutzmaßnahmen, wie digitale Signatur und Verschlüsselung, auf XML-Dokumente angewendet werden können. Im Anschluss wird noch auf die Möglichkeiten eingegangen die XML-Firewalls bieten.

2.7.1 XML-Signature

XML Signature (XMLdsig) [111] ist eine Empfehlung aus dem Jahr 2002, die durch die Arbeitsgruppe *IETF/W3C XML Signature Working Group* (Internet Engineering Task Force/ World Wide Web Consortium) erarbeitet wurde. Die Empfehlung ist auch unter RFC 3275 bekannt und beschreibt die Regeln und den Syntax, um digitale Signaturen in XML-Dokumente einzubetten. XMLdsig beschreibt keinen völlig neuen Weg um Dokumente digital zu signieren. Schon vorher war es möglich XML-Dokumente zu signieren, jedoch noch nicht eine Signatur im XML-Format auszudrücken oder einzelne Teile von XML-Dokumenten zu signieren. Dies wird nun durch Einsatz von XMLdsig ermöglicht. XMLdsig kann jedoch nicht nur auf XML-Dokumente angewendet werden, sondern auch auf beliebige andere Daten.
Um digitale Signaturen auf XML-Dokumente anwenden zu können, ist es notwendig, auf eine besondere XML-Eigenschaft einzugehen. XML-Dokument semantisch gleichen Inhalts können laut Spezifikation unterschiedlich repräsentiert werden. So sind beispielsweise beliebig viele Leerzeichen in einem Element-Tag einfügbar oder die Reihenfolge von Attributen beliebig änderbar, ohne dass sich dadurch die Bedeutung des Dokuments verändert. Damit existieren für ein und dasselbe XML-Dokument mehrere Darstellungen. Dies ist jedoch bei der Überprüfung von digitalen Signaturen problematisch. Wie bereits erwähnt, wird zur Erstellung von digitalen Signaturen zunächst ein Hash-Wert des Dokuments berechnet. Die dafür angewendete Hash-Funktion ist absichtlich so konstruiert, dass sie selbst im Falle geringster Änderung des Dokuments einen anderen Wert berechnet. Auf Empfangsseite muss also das XML-Dokument für die Überprüfung des Hash-Werts genau in derselben Repräsentation vorliegen wie auf Senderseite. Um dieser Problematik zu begegnen hat das W3C im Jahr 2001 die *Canonical XML* Empfehlung [108] veröffentlicht. Die Empfehlung erlaubt es, XML-Dokumente in eine einheitliche Darstellung, eine so genannte kanonische Darstellung zu überführen. Dazu werden beispielsweise überflüssige Leerzeichen entfernt und Attribute lexikalisch sortiert. Dieser Prozess wird als *Kanonisierung* bezeichnet. Dadurch können Hash-Werte auf einer einheitlichen Darstellung eines XML-Dokuments berechnet werden und somit die digitale Signatur sinnvoll eingesetzt werden.
Ein weiterer wichtiger Aspekt bei der Anwendung von XMLdsig sind Transformationen. Transformationen werden zur Umwandlung von referenzierten Daten und zur gezielten Auswahl von Teilbäumen eines XML Dokuments vor der Signaturberechnung angewendet. Die XMLdsig Empfehlung nennt (vgl. [111]) die Transformationsmechanismen *Canonicalization* (Kanonisierung), *Base64* [31], *XPath Filtering* [106], *Enveloped Signature Transform* (diese entfernt ein `<Signature>`-Element vor der Berechnung eines Hash-Werts) und *XSLT Transform* [107].

```
 1: <Signature ID?>
 2:  <SignedInfo>
 3:    <CanonicalizationMethod/>
 4:    <SignatureMethod/>
 5:    (<Reference URI?>
 6:      (<Transforms>)?
 7:      <DigestMethod>
 8:      <DigestValue>
 9:    </Reference>)+
10:  </SignedInfo>
11:  <SignatureValue>
12: (<KeyInfo>)?
13: (<Object ID?>)*
14: </Signature>
```

Abbildung 2.27: Genereller Aufbau von XML-Signature

In Abbildung 2.27 (vgl. [111]) ist der generelle Aufbau einer XML-Signatur dargestellt. Ein Fragezeichen beschreibt in dieser Struktur, dass ein Element bzw. Attribut keinmal oder einmal auftreten kann. Durch das Pluszeichen wird ein einmaliges oder mehrmaliges Auftreten gekennzeichnet und ein Sternchen erlaubt es, dass ein Element keinmal, einmal oder mehrmals auftreten kann. Das Element `<Signature>` repräsentiert das Wurzelelement einer XML-Signatur. Es kann das Attribut `ID` beinhalten, dass der Signatur eine eindeutige Identifikation vergibt. Dies ist sinnvoll, wenn das Dokument mehrere Signaturen enthält und Signaturen referenziert werden sollen. Darin verschachtelt befinden sich die Elemente `<SignedInfo>`, `<SignatureValue>` und optional die Elemente `<KeyInfo>` und `<Object>`.

`<SignedInfo>` ist ein Pflichtelement und beschreibt die Informationen, die signiert werden sollen. Darin verschachtelt sind die Pflichtelemente `<CanonicalizationMethod>` und `<SignatureMethod>`. Diese beschreiben einerseits den verwendeten Kanonisierungsalgorithmus und andererseits den Algorithmus, der zur Erstellung und Validierung der Signatur verwendet wird. `<SignatureMethod>` identifiziert damit alle kryptografischen Funktionen (Hash-Funktionen, Chiffrierungsalgorithmen, ...), die im Zusammenhang mit der Signatur verwendet werden.

Das `<SignedInfo>`-Element muss zudem mindestens einmal das Element `<Reference>` enthalten. Dieses gibt an, welche Daten in die Signaturberechnung mit einbezogen werden. Für jede zu signierende Ressource muss ein entsprechendes `<Reference>`-Element angegeben werden. Das optionale `URI`-Attribut gibt an, wo das zu signierende Objekt zu finden ist. In Abbildung 2.28 Zeile 8 (vgl. [111]) ist hier eine Referenz auf eine externe XML-Datei angegeben. Das `URI`-Attribut kann jedoch auch auf ein Element innerhalb dieses Dokuments verweisen. Dazu muss dieses Element mit einem `ID`-Attribut versehen werden. Die Referenz verweist dann auf dieses Element mit Attributen der Form `URI="#ElementID"`. Das optionale Kindelement `<Transforms>` gibt die vor der Signierung evtl. durchzuführenden Transformationen an. Mit dem `<DigestMethod>`-Element wird der Algorithmus angegeben, der zur Berechnung des Hash-Werts verwendet wird. `<DigestValue>` enthält dann den über die jeweiligen Ressourcen berechneten Hash-Wert.

Das Kindelement `<SignatureValue>` enthält die über das `<SignedInfo>`-Element berechnete Signatur. Diese wird in Base64-Kodierung [31] angegeben.

Optional enthält das `<Signature>`-Element das Element `<KeyInfo>`. Dieses wird dazu ver-

```
 1: <Signature Id="MyFirstSignature"
 2:   xmlns="http://www.w3.org/2000/09/xmldsig#">
 3:   <SignedInfo>
 4:     <CanonicalizationMethod
 5:      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
 6:     <SignatureMethod
 7:      Algorithm"http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
 8:     <Reference URI="http://www.einewebadresse.com/data.xml">
 9:       <Transforms>
10:         <Transform Algorithm=
11:           "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
12:       </Transforms>
13:       <DigestMethod Algorithm=
14:         "http://www.w3.org/2000/09/xmldsig#sha1"/>
15:       <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
16:    </Reference>
17:   </SignedInfo>
18:   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
19: </Signature>
```

Abbildung 2.28: Beispiel einer XML-Signatur

wendet, dem Empfänger des signierten XML-Dokuments anzuzeigen, welcher Schlüssel oder welche Zertifikate zur Überprüfung der Signatur benötigt werden. Dieses Element kann weggelassen werden, wenn der Empfänger bereits Kenntnis über diese Information hat. Mögliche Kindelemente sind hier z. B. `<KeyName>`, `<KeyValue>`, `<X509Data>` und `<PGPData>`. In Abbildung 2.29 ist eine Signatur mit der Verwendung eines `<KeyInfo>`-Elements dargestellt. Das `<KeyValue>`-Element enthält hier ein `<RSAKeyValue>`-Element, das den öffentlichen Schlüssel eines RSA-Kryptosystems repräsentiert.

Das weitere optionale `<Object>`-Element kann beliebige Daten enthalten. In der Praxis werden hier meist nur die Elemente `<Manifest>` und `<SignatureProperties>` verwendet. Das `<Manifest>`-Element kann anwendungsspezifische `<Reference>`-Elemente enthalten, die den gleichen Aufbau haben wie die `<Reference>`-Elemente des `<SignedInfo>`-Tags. Mit dem `<SignatureProperties>`-Element können zusätzliche Informationen, wie beispielsweise der Zeitpunkt der Erstellung der Signatur, angegeben werden. Zudem findet das `<Object>`-Element Verwendung in einer der im Folgenden beschriebenen drei Signaturvarianten.

XMLdsig unterscheidet drei verschiedene Signaturvarianten. Diese beschreiben die Art wie die Signatur an die zu signierenden Daten gebunden werden kann. Ist die Signatur in dem zu signierenden Dokument selbst abgelegt, so wird diese entweder als „*enveloped signature*“ (eingebettete Signatur) oder als „*enveloping signature*“ (einbettende Signatur) bezeichnet. Des Weiteren wird eine Signatur als „*detached signature*“ (losgelöste Signatur) bezeichnet, wenn die zu signierenden Daten nicht Teil des XML-Dokuments sind. In Abbildung 2.28 ist eine „*detached signature*“ dargestellt. In Zeile 8 ist hier eine Referenz auf ein externes Dokument angegeben. Abbildung 2.30 zeigt den Unterschied zwischen einer „*enveloped signature*“ und einer „*enveloping signature*“.

```
 1: <Signature Id="MyFirstSignature"
 2:   xmlns="http://www.w3.org/2000/09/xmldsig\#">
 3:   <SignedInfo>...</SignedInfo>
 4:   <SignatureValue>...</SignatureValue>
 5:   <KeyInfo>
 6:     <KeyValue>
 7:       <RSAKeyValue>
 8:         <Modulus>
 9:           xA7SEU+e0yQH5rm9kbCDN9o3aPIo7HbP7tX6W0
10:           ocLZAtNfyxSZDU16ksL6WjubafOgNEpcwR3RdF
11:           sT7bCgnXPBe5ELh5u4VEy19MzxkXRgrMvavzyB
12:           pVRgBUwU1V5foK5hhmbktQhyNdy/6Lp5foK5hh
13:           mbk5foK5hhmbktQhyNdy/61-pQRhDUDsTvK+g9
14:           cj47es9AQJ3U==
15:         </Modulus>
16:         <Exponent>AQAB</Exponent>
17:       </RSAKeyValue>
18:     </KeyValue>
19:   </KeyInfo>
20: </Signature>
```

Abbildung 2.29: XML-Signature mit Angabe eines RSA-Schlüssels

2.7.2 XML-Encryption

XML Encryption (XMLenc) [110] wurde im Jahr 2002 von der *W3C XML Encryption Working Group* als Empfehlung veröffentlicht. Die Spezifikation beschreibt den Syntax und die Vorgehensweise wie verschlüsselte Daten in XML dargestellt werden können und wie Teile eines XML-Dokuments selektiv verschlüsselt werden können. Dazu spezifiziert XMLenc das Element `<EncryptedData>`, dessen generelle Struktur in Abbildung 2.31 dargestellt ist.

Die Verschlüsselung wird bei XMLenc also durch ein `<EncryptedData>`-Element repräsentiert. Dieses Element kann beliebig oft innerhalb eines XML-Dokuments enthalten sein. Dadurch ist beispielsweise möglich, verschiedene Dokumententeile mit unterschiedlichen Schlüsseln zu chiffrieren. Das `<EncryptedData>`-Element darf jedoch nicht mit sich selbst verschachtelt sein. Dieses Element kann mehrere optionale Attribute enthalten. Das `ID`-Attribut ist bereits aus XMLdsig bekannt und dient auch hier zur Vergabe eines Identifikators und zu einer späteren Referenzierung des Elements. Das Attribut `Type` gibt den Verschlüsselungstyp an. Hier wird beispielsweise zwischen `Element` und `Content` unterschieden. Die Angabe des `MimeType`-Attributs kommt zum Einsatz, wenn beliebige Daten verschlüsselt werden sollen. In diesem Fall kann mit dem `Encoding`-Attribut auch die Kodierung (z. B. Base64) der Daten angegeben werden.

`<EncryptionMethod>` stellt ein optionales Kindelement von `<EncryptedData>` dar und definiert mit dem Attribut `Algorithm` den bei der Verschlüsselung verwendeten Algorithmus. Über weitere Kindelemente können beispielsweise Informationen über die verwendete Schlüssellänge (Attribut `KeySize`) angegeben werden. Wird dieses Element weggelassen, muss dem Empfänger der verwendete Algorithmus und dessen Parameter bekannt sein.

In Abbildung 2.32 ist die Verwendung des `<EncryptionMethod>`-Elements beispielhaft dargestellt. In diesem Beispiel wird ein 3DES-Algorithmus verwendet. Zudem wird die Verwendung des im Folgenden beschriebenen `<ds:KeyInfo>`-Elements gezeigt. Hier wird auf

```
Enveloped Signature:
1: <DocumentRoot ID="id1">
2:   ...
3:   <Signature>
4:     <SignedInfo>
6:       <Reference URI="#id1" />
7:     </SignedInfo>
8:   </Signature>
9: </DocumentRoot>

Enveloping Signature:
1: <Signature>
2:   <SignedInfo>
3:     <Reference URI="#id1" />
4:   </SignedInfo>
5:   <Object ID="id1">
6:     Eingeschlossenes XML-Dokument
7:   </Object>
8: </Signature>
```

Abbildung 2.30: Enveloped- und Enveloping XML-Signature

einen Schlüssel mit dem symbolischen Namen `Key A` verwiesen, der dem Empfänger der verschlüsselten Nachricht unter diesem Namen bekannt sein sollte.

Das `<ds:KeyInfo>`-Element ist bereits aus dem Namensraum von XMLdsig bekannt und dient auch hier zur Angabe von Informationen über den verwendeten Schlüssel. Dieses Element ist optional. Das `<ds:KeyInfo>`-Element hat die gleichen optionalen Kindelemente wie bereits in XMLdsig definiert. Dazu gehören beispielsweise die Elemente `<ds:KeyName>` und `<ds:RetrievalMethod>`. `<ds:RetrievalMethod>` gibt den Typ des Schlüssels und wo dieser zu finden ist an. Das `<ds:KeyName>`-Element stellt eine alternative Methode dar, um den Schlüssel für die Dechiffrierung zu identifizieren. XMLenc erweitert `<ds:KeyInfo>` um die zusätzlichen Elemente `<AgreementMethod>` und `<EncryptedKey>`. Die Angabe der `<AgreementMethod>` ist optional und dient der Schlüsselvereinbarung zwischen Sender und Empfänger. Der verwendete Schlüssel wird hier anhand von einem Geheimnis und einem festgelegten Algorithmus abgeleitet.

Das `<EncryptedKey>`-Element hat einen ähnlichen Aufbau wie das `<EncryptedData>`-Element, da beide von dem in der Spezifikation definierten abstrakten Typ `<EncryptedType>` abgeleitet sind. `<EncryptedKey>` beschreibt den verwendeten Schlüssel. Über das Kindelement `<ReferenceList>` werden die Elemente des XML-Dokuments angegeben, die mit diesem Schlüssel chiffriert wurden (vgl. Abbildung 2.33 Zeile 20-22). Dies können Referenzen auf `<EncryptedKey>`- oder `<EncryptedData>`-Elemente sein, die durch die beiden Kindelemente `<KeyReference>` bzw. `<DataReference>` repräsentiert werden.

Im vorherigen Beispiel wurde der Schlüssel mittels der Angabe eines symbolischen Namens bekannt gemacht. Es ist jedoch auch möglich den Schlüssel direkt im Dokument abzulegen. Dazu sollte der Schlüssel natürlich selbst auch verschlüsselt werden. Hierzu kann ein asymmetrisches Verfahren angewendet werden. Das in Abbildung 2.33 dargestellte Beispiel zeigt die Kombination eines symmetrischen und asymmetrischen Verfahrens und verdeutlicht damit die Verwendung eines hybriden Verschlüsselungsverfahrens. Das `<ds:KeyInfo>`-Element (Zeile

```
 1: <EncryptedData Id? Type? MimeType? Encoding?>
 2:   <EncryptionMethod/>?
 3:   <ds:KeyInfo>
 4:     <EncryptedKey>?
 5:     <AgreementMethod>?
 6:     <ds:KeyName>?
 7:     <ds:RetrievalMethod>?
 8:     <ds:*>?
 9:   </ds:KeyInfo>?
10:   <CipherData>
11:     <CipherValue>?
12:     <CipherReference URI?>?
13:   </CipherData>
14:   <EncryptionProperties>?
15: </EncryptedData>
```

Abbildung 2.31: Genereller Aufbau von XML-Encryption

```
 1: <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
 2:   Type="http://www.w3.org/2001/04/xmlenc\#Element"/>
 3:   <EncryptionMethod Algorithm=
 4:     http://www.w3.org/2001/04/xmlenc\#tripledes-cbc"/>
 5:   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig\#">
 6:     <ds:KeyName>Key A</ds:KeyName>
 7:   </ds:KeyInfo>
 8:   <CipherData>
 9:     <CipherValue>A23B45C56</CipherValue>
10:   </CipherData>
11: </EncryptedData>
```

Abbildung 2.32: Verschlüsselung mit Angaben des Schlüssels und der Verschlüsselungsmethode

5) verweist in diesem Beispiel auf einen Schlüssel, der weiter unten im Dokument angegeben ist (Zeile 11-23). Dieser ist, wie an den Zeilen 13 und 14 zu erkennen ist, mittels des asymmetrischen RSA-Verfahrens verschlüsselt.
Das `<CipherData>`-Element ist ein erforderliches Kindelement von `<EncryptedData>` und enthält entweder direkt die verschlüsselten Daten unter Angabe des `<CipherValue>`-Elements oder eine Referenz (`<CipherReference>`) auf eine externe Ressource.
Für zusätzliche Informationen, wie zum Beispiel der Zeitpunkt der Erstellung der Verschlüsselung, Angaben über das Verschlüsselungssystem selbst oder andere anwendungsspezifische Daten, steht noch das optionale Element `<EncryptionProperties>` zur Verfügung. Es erlaubt diese Informationen für die Elemente `<EncryptedData>` und `<EncryptedKey>`.
XMLenc erlaubt es, das komplette XML-Dokument oder nur bestimmte Teilbäume davon zu verschlüsseln. Auf welcher Ebene die Verschlüsselung erfolgt, bestimmt die Granularität der Verschlüsselung. Die Verschlüsselung sollte natürlich alle Daten beinhalten, die schützenswert sind, jedoch auch nicht mehr Daten als nötig, um eine schnelle und effiziente Verarbeitung zu gewährleisten.
XMLenc unterscheidet hierfür grundsätzlich vier verschiedene Verschlüsselungsarten. Zur Veranschaulichung der unterschiedlichen Granularität der Verschlüsselung soll das Beispiel aus der

```
 1: <EncryptedData Id="SecretData"
 2:   xmlns="http://www.w3.org/2001/04/xmlenc\#">
 3:   <EncryptionMethod Algorithm=
 4:     "http://www.w3.org/2001/04/xmlenc\#tripledes-cbc"/>
 5:     <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig\#">
 6:       <ds:RetrievalMethod URI="\#KeyA"
 7:       Type="http://www.w3.org/2001/04/xmlenc\#EncryptedKey"/>
 8:     </ds:KeyInfo>
 9:     <CipherData><CipherValue>dfwesd</CipherValue></CipherData>
10: </EncryptedData>
11: <EncryptedKey Id="KeyA"
12:   xmlns="http://www.w3.org/2001/04/xmlenc\#">
13:   <EncryptionMethod Algorithm=
14:     "http://www.w3.org/2001/04/xmlenc\#rsa-1_5"/>
15:   <ds:KeyInfo
16:     xmlns:ds="http://www.w3.org/2000/09/xmldsig\#">
17:     <ds:KeyName>Key B</ds:KeyName>
18:   </ds:KeyInfo>
19:   <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
20:   <ReferenceList>
21:     <DataReference URI="\#SecretData"/>
22:   </ReferenceList>
23: </EncryptedKey>
```

Abbildung 2.33: Hybride Verschlüsselung

Spezifikation [110] dienen, dargestellt in Abbildung 2.34.

```
1: <PaymentInfo>
2:   <Name>John Smith</Name>
3:   <CreditCard Limit="5,000" Currency="USD">
4:     <Number>4019 2445 0277 5567</Number>
5:     <Issuer>Example Bank</Issuer>
6:     <Expiration>04/02</Expiration>
7:   </CreditCard>
8: </PaymentInfo>
```

Abbildung 2.34: Ein XML-Dokument als Ausgangsbeispiel für eine XML-Verschlüsselung

- Verschlüsselung von beliebigen Daten

  XMLenc erlaubt es, beliebige Daten zu verschlüsseln. Dazu können beispielsweise Bilder, PDF-Dokumente oder auch XML-Dokumente selbst gehören. Durch Angabe des `MimeType`-Attributs innerhalb des `<EncryptedData>`-Elements kann der Typ der Daten spezifiziert werden.

  Wenn das gesamte Dokument aus Abbildung 2.34 verschlüsselt werden soll, so wird das komplette Wurzelelement `<PaymentInfo>` durch ein `<EncryptedData>`-Element ersetzt. Das resultierende XML-Dokument ist in Abbildung 2.35 dargestellt.

- Verschlüsselung eines XML-Elements

```
1: <?xml version="1.0"?>
2: <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
3:   MimeType="text/xml">
4:   <CipherData>
5:     <CipherValue>A23B45C56</CipherValue>
6:   </CipherData>
7: </EncryptedData>
```

Abbildung 2.35: XML-Encryption des kompletten Dokuments

In dem Beispiel kann es jedoch auch sinnvoll sein, wenn beispielsweise der Name für eine interne Zuordnung der Zahlung benötigt wird, nur die Informationen über die Zahlungsmethode zu verschlüsseln. In diesem Fall wird nur der `<CreditCard>`-Tag durch das Element `<EncryptedData>` ersetzt (vgl. Abbildung 2.36).

Bei der Verschlüsselung eines XML-Elements wird also das Element inklusive seiner Attribute und allen Kindelementen durch das Element `<EncryptedData>` ersetzt. Das Attribut `Type` hat in diesem Fall den Wert `Element` (vgl. Abbildung 2.36 Zeile 4). Diese Verschlüsselung wird daher auch als eine Verschlüsselung auf *Element*-Ebene bezeichnet.

```
1: <PaymentInfo>
2:   <Name>John Smith</Name>
3:    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
4:     Type="http://www.w3.org/2001/04/xmlenc\#Element">
5:      <CipherData>
6:       <CipherValue>A23B45C56</CipherValue>
7:     </CipherData>
8:   </EncryptedData>
9: </PaymentInfo>
```

Abbildung 2.36: XML-Encryption auf Element-Ebene

- Verschlüsselung des Inhalts eines XML-Elements

  Bei der Verschlüsselung des Inhalts eines XML-Elements bleiben das Element und seine Attribute im Klartext vorhanden, der Inhalt (Text oder weitere Elemente) wird jedoch ersetzt. Die Verschlüsselung erfolgt hier auf Inhaltsebene (*Content*-Ebene). In dem Beispiel kann es notwendig sein, dass einzelne Instanzen in der Verarbeitungskette wissen müssen, mit welcher Methode die Zahlung erfolgt. Die genauen Daten der Kreditkarte müssen jedoch vertrauenswürdig behandelt werden. Hierfür kann nun die beschriebene Granularität der Verschlüsselung eingesetzt werden. In diesem Fall hat das `Type`-Attribut des `<EncryptedData>`-Elements den Wert `Content`. Dies ist in Abbildung 2.37 dargestellt.

- Super-Encryption - Verschlüsselung von verschlüsselten Daten

  Die in der Spezifikation genannte *Super-Encryption* erlaubt es, bereits verschlüsselte Daten erneut zu verschlüsseln. Hier muss jedoch beachtet werden, dass bei der Mehrfachverschlüsselung der Elemente `<EncryptedData>` oder `<EncryptedKey>` immer nur das komplette Element verschlüsselt werden kann. Ein Herunterbrechen der Granularität auf Inhaltsebene ist nicht gestattet.

```
 1: <PaymentInfo>
 2:   <Name>John Smith</Name>
 3:   <CreditCard Limit="5,000" Currency="USD">
 4:     <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc\#"
 5:       Type="http://www.w3.org/2001/04/xmlenc\#Content">
 6:       <CipherData>
 7:         <CipherValue>A23B45C56</CipherValue>
 8:       </CipherData>
 9:     </EncryptedData>
10:   </Creditcard>
11: </PaymentInfo>
```

Abbildung 2.37: XML-Encryption auf Content-Ebene

Mit der Mehrfachverschlüsselung lassen sich hierarchische Zugriffsrechte realisieren. Beispielsweise könnte ein `<EncryptedData>`-Element die Personaldaten eines Mitarbeiters verschlüsseln. In den Personaldaten selbst sind neben den Mitarbeiterdaten wie Name, Adresse, etc. auch weitere verschlüsselte Daten, wie beispielsweise Informationen über das Gehalt oder die Beurteilung des Mitarbeiters, enthalten. Instanzen, die den passenden Schlüssel für die erste Ebene besitzen, können zwar auf die Mitarbeiterdaten zugreifen, jedoch nicht die Informationen zu Gehalt und Beurteilung des Mitarbeiters einsehen. Diese können nur von Instanzen eingesehen werden, die den dazu passenden Schlüssel besitzen.

2.7.3 XML-Firewalls

Die Sicherheit von XML-Anwendungen und damit von Web Services kann maßgeblich durch den Einsatz von so genannten *XML-Firewalls* (*Web Service Firewalls*, *Web Service Security-Gateways*, *SOAP-Proxies*) erhöht werden. Bevor die Besonderheiten einer XML-Firewall untersucht werden, wird im Folgenden kurz erläutert, was generell unter einer Firewall zu verstehen ist.

Als Firewall werden Systeme aus Hardware- und Softwarekomponenten bezeichnet, die den Zugriff zwischen verschiedenen Rechnernetzen reglementieren, in dem sie ein definiertes Sicherheitskonzept umsetzen. Firewalls werden im Allgemeinen zwischen dem unsicheren Internet und einem sicheren Intranet eingesetzt und schützen das Intranet vor unberechtigten Zugriffen von außen und reglementieren gleichzeitig den Zugriff vom Intranet auf das Internet.

Grundsätzlich werden zwei unterschiedliche Technologien zum Aufbau einer Firewall unterschieden. *Paketfilter-Firewalls* analysieren auf Netzwerk- bzw. Transportebene die ankommenden und weggehenden TCP- und UDP-Pakete und entscheiden anhand von Header Informationen, ob ein Paket die Firewall passieren darf oder geblockt wird. Sie arbeiten mit Gebots- und Verbotsregeln, die von einem Administrator definiert werden.

Application Layer Firewalls arbeiten dagegen auf Anwendungsebene. Sie ermöglichen es, mit Hilfe von so genannten Content-Filtern und *Proxies* die Nutzdaten einer Anwendung zu analysieren und ggf. unerwünschte Inhalte, wie beispielsweise Viren und Spam-Mails, zu entfernen. Ein *Proxy* ist ein Stellvertreter für den Client im Intranet. Er baut für diesen stellvertretend die Verbindung zu Servern im Internet auf. Application Layer Firewalls bieten zudem meist auch Authentifikationsmechanismen an.

Da Web Services meist den Port 80 für die Kommunikation nutzen und dieser grundsätzlich von einer Paketfilter-Firewall nicht untersucht wird, muss es auf Anwendungsebene eine Siche-

rungsmöglichkeit geben. Hierzu werden XML-Firewalls eingesetzt, die ähnlich wie Application Layer Firewalls arbeiten, jedoch auf Anwendungen, die Protokolle auf XML-Basis einsetzen, spezialisiert sind.
Sie bieten Sicherheitsmechanismen zur Realisierung von Authentifikation, Autorisierung und Protokollierung an und ermöglichen es, XML-Nachrichten zu validieren, Header zu überprüfen und Sicherheitsrichtlinien durchzusetzen. XML-Firewalls schützen damit Web Service-Anwendungen und XML-Parser vor Bedrohungen aus dem Internet.
Dazu prüfen sie beispielsweise die ankommenden XML-Nachrichten auf ihre Konformität zu einem definierten XML-Schema. Damit werden ungültige Werte vorzeitig erkannt und Web Services vor Angriffen wie Pufferüberläufen geschützt. XML-Firewalls können in der Regel auch digitale Signaturen überprüfen und auf dieser Basis eine Authentifikation und Autorisierung von Zugriffen vornehmen.
XML-Firewalls arbeiten auch mit Hilfe heuristischer Mechanismen. Beispielswiese können sie Statistiken über die Größe einer ankommenden Anfrage an einen Web Service speichern. Kommt es nun vor, dass eine Web Service eine Anfrage erhält, die ungewöhnlich groß ist, so können vordefinierte Aktionen eingeleitet werden. Die XML-Firewall kann in diesem Fall beispielsweise detailliertere Protokollierungen vornehmen, einen Administrator automatisch benachrichtigen oder die Anfrage verwerfen.
XML-Firewalls werden derzeit von vielen verschiedenen Herstellern angeboten und ständig weiterentwickelt und verbessert.

2.8 Web Service-Sicherheit

Die vorgestellten Standards *XMLenc* und *XMLdsig* ermöglichen es, XML-Dokumente im Allgemeinen zu verschlüsseln bzw. zu signieren. Wie diese Lösungen jedoch innerhalb der Web Services-Welt eingesetzt werden können, ist noch nicht festgelegt. Aus diesem Grund wurde im April 2002 von Microsoft, IBM und VeriSign die *Web Service Security*-Spezifikation veröffentlicht. Die Spezifikation beschreibt, auf Basis der Erweiterbarkeit von SOAP, wie die Integrität und Vertraulichkeit von Nachrichten und der Austausch von Sicherheitsinformationen mit Web Services ermöglicht wird. Zudem wurde der Fahrplan „Security in a Web Services World: A Proposed Architecture and Roadmap“ [57] veröffentlicht, der eine Sicherheitsstrategie für Web Services auf verschiedenen Ebenen vorschlägt.
Im Folgenden werden daher die in diesem Fahrplan vorgeschlagenen WS*-Spezifikationen detailliert vorgestellt. Anschließend werden noch weitere WS*-Spezifikationen und einige wichtige Spezifikationen beschrieben, die im Web Services-Umfeld für verschiedene Sicherheitsaspekte zum Einsatz kommen können. Zunächst wird jedoch noch kurz auf die Sicherheitsmechanismen eingegangen, die auf der Transportebene angewendet werden können.

2.8.1 Sicherheitsmechanismen auf Transportebene

Das wichtigste Sicherheitsprotokoll auf Transportebene ist das *Secure Socket Layer (SSL)* Protokoll bzw. sein Nachfolger, das *Transport Layer Security (TLS)* Protokoll. SSL wurde ursprünglich von dem Unternehmen Netscape im Jahr 1994 entwickelt und im Jahr 1999 als RFC2246 [27] veröffentlicht und in TLS umbenannt. Seit April 2006 liegt die Version 1.1 als RFC4346 [32] vor.
SSL/TLS wird vorwiegend auf Webseiten eingesetzt, wenn eine sichere Verbindung zwischen dem Computer des Benutzers und der Seite des Anbieters benötigt wird, um beispielsweise die Bestellung von Produkten sicher abwickeln zu können oder Kreditkarteninformationen sicher

übermitteln zu können. SSL/TLS verwendet dazu ein *Handshake*-Verfahren, um die Parameter der symmetrischen Verschlüsselungsform zu vereinbaren und einen Sitzungsschlüssel auszutauschen. Dies wird über das *SSL-Handshake-Protocol* abgebildet. Zudem wird über dieses Protokoll auch die Authentifikation von Servern gegenüber Clients mittels eines asymmetrischen Kryptosystems abgewickelt. Optional kann auch ein Client aufgefordert werden, sich gegenüber des Servers zu authentifizieren. Die symmetrische Verschlüsselung und Sicherung der Integrität durch Berechnung von kryptografischen Hash-Werten wird mittels des *SSL-Record-Protocol* durchgeführt.

SSL/TLS ermöglicht es damit die Schutzziele, Vertraulichkeit und Integrität auf Transportebene zwischen zwei Endpunkten zu gewährleisten und außerdem Server und Client wechselseitig zu authentifizieren. Für die Absicherung von Web Services kann SSL nur bedingt eingesetzt werden. Zwar lässt sich der Transportweg absichern, jedoch nicht das Dokument selbst. Eine selektive Verschlüsselung oder Signatur der Inhalte einer SOAP-Nachricht ist nicht möglich.

Zudem ist mit SSL/TLS nur eine Punkt-zu-Punkt-Sicherheit zu gewährleisten, eine Ende-zu-Ende-Sicherheit jedoch nicht. Spätestens, wenn eine abgeschickte SOAP-Nachricht nicht direkt zum endgültigen Empfänger gelangt, sondern zunächst zu einem *SOAP-Intermediary* gesendet wird, kann mit SSL/TLS die Sicherheit nicht mehr gewährleistet werden. Auf der Zwischenstation liegt die Nachricht dann im Klartext vor und stellt somit ein Angriffsziel dar. Außerdem kann die Zwischenstation Nachrichtenteile einsehen, die evtl. nicht für sie bestimmt sind. Hier wird also eine selektive Verschlüsselung bzw. Signatur benötigt.

Neben SSL/TLS existieren noch weitere Standards, wie beispielsweise *IPSec* [26] oder *Secure Shell (SSH)* (vgl. RFC4250-4254 [60, 127, 125, 128, 126]), auf die hier jedoch nicht näher eingegangen wird. Sie besitzen ähnliche Nachteile, wenn sie zur Absicherung von Web Services eingesetzt werden sollen.

2.8.2 WS*

Der beschriebene Fahrplan bzw. die Roadmap zur Web Service-Sicherheit definiert verschiedene XML-Spezifikationen, die aufbauend auf der *Web Service Security*-Spezifikation unterschiedliche Aufgaben besitzen. Sie definieren wie so genannte *Security Tokens (ST)* ausgestellt werden, wie diese mit Identitäten verknüpft werden und wie Sicherheitsrichtlinien festgelegt und diese mit bestimmten Web Services verbunden werden.

Unter einem *Security Token (ST)* wird hier eine XML-Repräsentation von Sicherheitsinformationen verstanden. Diese können sowohl signiert, als auch unsigniert sein. Ein unsigniertes ST kann beispielsweise ein Passwort oder ein symmetrischer Schlüssel sein. Ein signiertes ST ist beispielsweise ein *X.509* oder ein Kerberos-Ticket.

Genauso wie Web Services eine Abstraktionsebene eingeführt haben, die es erlaubt, Anwendungen Plattform-, Programmiersprachen-, und Protokollunabhängigkeit zur Verfügung zu stellen, soll mit der vorgeschlagenen Roadmap eine Abstraktion auf der Sicherheitsebene möglich werden. Sie soll es erlauben, dass Unternehmen, die unterschiedliche Sicherheitstechnologien einsetzen, trotzdem sicher über SOAP miteinander kommunizieren können.

Abbildung 2.38 (vgl. [57]) zeigt die von Microsoft, IBM und VeriSign vorgeschlagene Roadmap. Auf unterster Ebene steht SOAP, gefolgt von der WS-Security-Spezifikation, die nun im Folgenden vorgestellt wird.

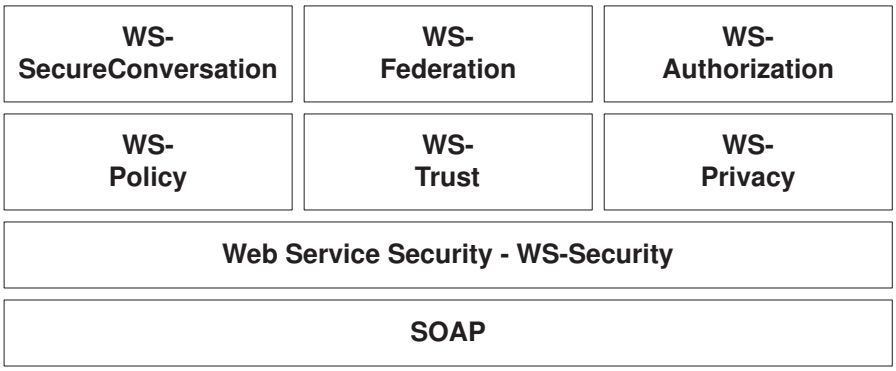

Abbildung 2.38: Web Services-Sicherheit Roadmap

2.8.3 WS-Security

Im Juli 2002 wurde die *Web Service Security (WS-Security, WSS)*-Spezifikation von Microsoft, IBM und VeriSign an die Organisation OASIS weitergegeben. Seitdem kümmert sich das *OASIS Web Services Security Technical Committee (WSS TC)* um die Weiterentwicklung der Sicherheitsaspekte im Web Services-Umfeld. Im Februar 2006 veröffentlichte dieses Komitee die WS-Security Version 1.1 [17] die folgende Teil-Spezifikationen umfasst:

- *WS-Security SOAP Message Security 1.1* [70]

  Diese Spezifikation bildet den Kern der WSS-Spezifikation. Sie definiert eine Menge von SOAP-Erweiterungen, die dazu verwendet werden können, die Integrität und Vertraulichkeit von SOAP-Nachrichten zu sichern. Zudem wird es dadurch möglich, Security Tokens als Teil einer SOAP-Nachricht zu versenden bzw. zu empfangen.

  Die Spezifikation ist sehr flexibel und kann als Basis für eine sichere Nutzung von Web Services innerhalb verschiedener Sicherheitsmodelle verwendet werden. Dafür enthält die Spezifikation verschiedene Profile, die die Nutzung von Security Tokens auf Basis von beispielsweise Kerberos, X.509 oder auch SAML ermöglicht.

  WS-Security kann keine Sicherheit garantieren oder gar eine vollständige Sicherheitslösung darstellen. Vielmehr dient es als Baustein, der zusammen mit anderen Spezifikationen und Protokollen zum Aufbau einer sicheren Architektur führen soll. Wie sicher diese ist, hängt massgeblich von den angewendeten kryptografischen Techniken ab.[70]

- *Username Token Profile 1.1* [71]

  Dieses Profil beschreibt wie ein `UsernameToken` innerhalb von WSS angewendet wird. Es definiert wie ein Subjekt identifiziert und im Kombination mit einem Passwort authentifiziert werden kann.

- *SAML Token Profile 1.1* [66]

  Das *SAML Token*-Profil definiert, wie *SAML-Assertions* innerhalb von WSS angewendet werden. Dazu spezifiziert es wie diese Assertions in das `<wsse:Security>`-Element eingebettet werden bzw. von diesem referenziert werden können. Zudem beschreibt es

[17] `http://www.oasis-open.org/specs/index.php#wssv1.1`

wie die Subjekte und Aussagen einer Assertion mit XMLdsig an eine SOAP-Nachricht gebunden werden können.

- *X.509 Token Profile 1.1* [72]

  Dieses Profil beschreibt die Verwendung von X.509 Zertifikaten innerhalb von WSS.

- *Kerberos Token Profile 1.1* [69]

  Das *Kerberos Token*-Profil beschreibt die Anwendung von Kerberos-Tokens innerhalb von WSS. Es definiert wie Kerberos-Tickets kodiert und in SOAP-Nachrichten eingebettet werden. Zudem spezifiziert es dessen Signatur und Verschlüsselung.

- *Rights Expression Language (REL) Token Profile 1.1* [15]

  Dieses Profil spezifiziert die Anwendung der *ISO/IEC 21000-5 Rights Expressions* [2] innerhalb von WSS.

- *SOAP with Attachments (SWA) Profile 1.1* [41]

  Dieses Profil beschreibt die Nutzung der *SOAP with Attachments*-Spezifikation[18] [4] innerhalb von WSS. Es spezifiziert wie die Integrität, Vertraulichkeit und Authentizität von SOAP-Attachments gewährleistet werden kann.

Innerhalb der WSS-Spezifikation werden bestehende Namensräume wiederverwendet sowie neue eingeführt. Die Tabelle 2.8 enthält eine Übersicht über diese Namensräume.

Prefix	Namensraum
`ds`	XMLdsig: `http://www.w3.org/2000/09/xmldsig#`
`xenc`	XMLenc: `http://www.w3.org/2001/04/xmlenc#`
`S11`	SOAP Version 1.1: `http://schemas.xmlsoap.org/soap/envelope/`
`S12`	SOAP Version 1.2: `http://www.w3.org/2003/05/soap-envelope`
`wsse`	WS-Security 1.0: `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`
`wsse11`	WS-Security 1.1: `http://www.oasis-open.org/committees/download.php/16791/oasis-wss-wssecurity-secext-1.1.xsd`
`wsu`	WS-Security Utility: `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`
`wsp`	WS-Policy & WS-PolicyAttachment: `http://schemas.xmlsoap.org/ws/2004/09/policy`
`sp`	WS-SecurityPolicy: `http://schemas.xmlsoap.org/ws/2005/07/securitypolicy`

[18] Die *SOAP with Attachments*-Spezifikation beschreibt, wie ein oder mehrere *Attachments* (Anhänge) in ihrem ursprünglichen Format in einer *Multipurpose Internet Mail Extensions (MIME)*-Struktur [25] an SOAP-Nachrichten angehängt werden können.

Prefix	Namensraum
wst	WS-Trust: http://schemas.xmlsoap.org/ws/2005/02/trust
wsc	WS-SecureConversation http://schemas.xmlsoap.org/ws/2005/02/sc
wsa	WS-Addressing http://schemas.xmlsoap.org/ws/2004/08/addressing

Tabelle 2.8: Web Service-Namensräume

Die WSS-Spezifikation definiert das <wss:Security>-Element als neues SOAP-Header Element. Es beinhaltet zwei Attribute. Das optionale actor-Attribut (SOAP 1.1) bzw. role-Attribut (SOAP 1.2) gibt an, an wen ein Security Token adressiert ist. Der Wert des Attributs wird durch eine der in der SOAP-Spezifikation definierten Rollen next, ultimateReceiver oder none bestimmt.
Mit dem Attribut mustUnderstand wird festgelegt, ob der Header vom Empfänger bearbeitet werden muss. Es kann die Werte „true“ bzw. „1“ oder „false“ bzw. „0“ annehmen. Wird das Attribut weggelassen, so ist dies semantisch mit dem Wert „false“ gleichzusetzen.
Der SOAP-Header kann mehrere <wss:Security>-Elemente beinhalten. Ist dies der Fall, so darf nur ein <wss:Security>-Element das Attribut role weglassen und es dürfen niemals mehrere Elemente mit dem gleichen role-Wert existieren. Ein <wss:Security>-Element ohne ein role-Attribut darf von jedem SOAP-Knoten bearbeitet werden, jedoch von diesen nicht entfernt werden. Jeder SOAP-Knoten kann <wss:Security>-Elemente hinzufügen bzw. die enthaltenen Informationen ergänzen. Wird das role-Attribut spezifiziert, so darf der adressierte Knoten das zugehörige Element auch entfernen.
Innerhalb des <wss:Security>-Elements können *XMLdsig*-, *XMLenc*- oder Security Token-Elemente eingefügt werden. Die Elemente von *XMLdsig* und *XMLenc* wurden bereits in Kapitel 2.7 vorgestellt. Nachfolgend wird noch aufgezeigt, wie diese Elemente innerhalb der WSS-Spezifikation angewendet werden. Zunächst werden jedoch die einzelnen Security Token Elemente vorgestellt.

2.8.3.1 WSS-Security Token

In der Spezifikation werden vier Security Tokens spezifiziert. Die Elemente UsernameToken, BinarySecurityToken, EncryptedData und SecurityTokenReference ermöglichen es, unterschiedliche Security Tokens in eine SOAP-Nachricht zu integrieren. Die Spezifikation erlaubt es zudem, eigene Element zu definieren.

UsernameToken Das <wsse:UsernameToken>-Element ermöglicht es, einfache Authentifikationsinformationen in den SOAP-Header aufzunehmen. Für die Authentifikation von Endbenutzern wird jedoch meist eine *SAML Assertion* oder ein *Kerberos Ticket* verwendet. Das Element kann jedoch als Basis für Gewährleistung der Authentizität einer SOAP-Nachricht verwendet werden. Für die spätere Referenzierung kann über das Element Id eine ID vergeben werden.
Das Element kann neben dem Benutzernamen im Element <wsse:Username> auch ein Passwort im Element <wsse:Password> aufnehmen. Das Attribut Type gibt in diesem Fall an, ob es sich um ein Passwort im Klartext (PasswordText) oder um den berechneten Hash-Wert eines Passworts (PasswordDigest) handelt. Die Elemente <wsse:Nonce>

und <wsu:Created> erlauben es, eine *Nonce* (eine Zufallszahl die nur einmal benutzt wird) und das Erstellungsdatum anzugeben, um so genannte Replay-Attacken (vgl. Kapitel 5.1.4) zu verhindern. Abbildung 2.39 zeigt den generellen Aufbau des <wsse:UsernameToken>-Elements (Zeilen 5-12), eingebettet in einer SOAP-Nachricht.[71]

```
 1: <Envelope xmlns:S11="..." xmlns:wsse="...">
 2:   <S11:Header>
 3:     ...
 4:     <wsse:Security>

 5:       <wsse:UsernameToken wsu:Id="Example-1">
 6:         <wsse:Username>...</wsse:Username>
 7:         <wsse:Password Type="...">...</wsse:Password>
 8:         <wsse:Nonce EncodingType="...">...</wsse:Nonce>
 9:         <wsu:Created>...</wsu:Created>
10:       </wsse:UsernameToken>

11:     </wsse:Security>
12:     ...
13:   </S11:Header>
14: </Envelope>
```

Abbildung 2.39: WSS-UsernameToken eingebettet in einer SOAP-Nachricht

Zur Absicherung der Integrität und Vertraulichkeit der angegebenen Informationen können sowohl digitale Signaturen als auch Verschlüsselungen verwendet werden. Zudem kann, wenn eine Punkt-zu-Punkt-Sicherheit ausreichend ist, auch eine Sicherung auf der Transportebene erfolgen.

BinarySecurityToken Mit einem <wsse:BinarySecurityToken>-Element können Informationen eingebettet werden, die nicht im XML-Format vorliegen. Dies können beispielsweise *X.509* Zertifikate oder *Kerberos-Tickets* sein. Auch hier kann über das Attribut Id eine ID zur späteren Referenzierung vergeben werden. Über die Attribute EncodingType und ValueType werden Informationen zur Interpretation des Inhalts des Elements angegeben. Das ValueType-Attribut gibt den Typ des Security Token an (z. B. ein X.509 Zertifikat der Version 3 mit dem Wert wsse:X509v3). Das Attribut EncodingType legt fest, in welchem Format das *Token* kodiert ist. Standardmäßig wird hier eine Base64-Kodierung verwendet. Abbildung 2.40 zeigt ein Base64-kodiertes *X.509* Zertifikat.

```
1: <wsse:BinarySecurityToken
2:    wsu:Id="MyCertificate"
2:    EncodingType="...#Base64Binary""
3:    ValueType="...#X509v3">
4: MIIEZzCCA9CgAwIBAgIQEmtJZc0hf739ut30t803z3
5: nf938HF139fvbfGtfCUvc427jF03eo2k1WDFhj2...
6: </wsse:BinarySecurityToken>
```

Abbildung 2.40: WSS X.509-BinarySecurityToken

EncryptedData In bestimmten Fällen ist es wünschenswert, ein Security Token in verschlüsselter Form in das `<wsse:Security>`-Element einzubetten. Dafür kann das aus der *XMLenc*-Spezifikation bekannte `<xenc:EncryptedData>`-Element verwendet werden. Dieses wurde bereits ausführlich in Kapitel 2.7.2 vorgestellt.

SecurityTokenReference Bei der Anwendung von XMLdsig oder XMLenc müssen Angaben über die verwendeten Schlüssel gemacht werden. Dies kann entweder direkt in der Nachricht oder außerhalb der Nachricht erfolgen. Das `<wsse:SecurityTokenReference>` Element stellt einen erweiterbaren Mechanismus dar, der zur Referenzierung von diesen Schlüsseln dient.
Das Attribut `Id` dient dazu, das Element zu referenzieren. Das optionale `TokenType` Attribut ermöglicht es, den Typ des referenzierten Security Token anzugeben. Die Spezifikation erlaubt es, über das Attribut `Usage` einen Wert, der die Verwendung des Tokens definiert, anzugeben. Diese können anwendungsspezifisch festgelegt werden.

```
1: <wsse:SecurityTokenReference
2:     wsu:Id="...", wsse11:TokenType="...",
3:     wsse:Usage="...">
4:   <wsse:Reference URI="..." ValueType="..."/>
5: </wsse:SecurityTokenReference>
```

Abbildung 2.41: Beispiel einer SecurityTokenReference

In Abbildung 2.41 ist der generelle Aufbau des `<wsse:SecurityTokenReference>`-Elements dargestellt. Innerhalb dieses Beispiels wird eine direkte Referenz auf ein Security Token angegeben. Die Spezifikation definiert dazu verschiedene Kindelemente für die Referenzierung von *Tokens*:

- *Direct References*

  Über das `<wsse:Reference>`-Element können direkte Referenzen angegeben werden. Mittels URI-Fragmenten wird auf *Tokens* innerhalb der gleichen SOAP-Nachricht verwiesen. Andernfalls kann eine komplette URI angegeben werden, die dann auf ein externes *Token* verweist.

- *Key Identifiers*

  Das `<wsse:KeyIdentifier>`-Element dient dazu, ein Security Token eindeutig zu identifizieren. Es ist vergleichbar mit dem `<ds:KeyName>`-Element der *XMLdsig*-Spezifikation, bietet jedoch die Möglichkeit, genauere Informationen über ein *Token* anzugeben. So kann auch der Typ und die Kodierung des *Tokens* angegeben werden.

- *ds:KeyName*

  Die Spezifikation erlaubt es, auch *Tokens* anhand von Zeichenketten durch die Verwendung des `<ds:KeyInfo>`-Elements der *XMLdsig*-Spezifikation zu referenzieren, empfiehlt jedoch stattdessen die Verwendung des `<wsse:KeyIdentifier>`-Elements.

- *Embedded References*

  Eine so genannte *Embedded Reference* dient dazu, ein Security Token direkt in das Element `SecurityTokenReference` einzubinden.

- *ds:KeyInfo*

  Auch das <ds:KeyInfo>-Element der XMLdsig-Spezifikation kann zur Referenzierung von Security Tokens verwendet werden.

2.8.3.2 Digitale Signaturen

Zur Signierung von SOAP-Nachrichten wird der existierende Standard XMLdsig wiederverwendet. Die WSS-Spezifikation erlaubt es, mehrere, jeweils unterschiedliche Teile, die sich sogar überschneiden können, einer SOAP-Nachricht zu signieren. Dazu werden jeweils mehrere <ds:Signature>-Elemente innerhalb eines oder ggf. auch mehreren <wsse:Security>-Elementen in eine SOAP-Nachricht eingefügt. Dies ist besonders für Nachrichten wichtig, die mehrere Zwischenknoten passieren, bevor sie ihr endgültiges Ziel erreichen. Jeder Zwischenknoten kann anhand der Signatur die Integrität und ggf. Authentizität überprüfen und nach den vorgenommenen Änderungen die Integrität und Authentizität der Informationen für den nächsten Knoten sicherstellen, in dem er seinerseits eine Signatur einfügt.
Werden Signaturen und Verschlüsselungen gemeinsam verwendet, so müssen die entsprechenden Elemente <ds:Signature> und <xenc:EncryptedData> in einer vordefinierten Reihenfolge in das <wsse:Security>-Element eingefügt werden. Damit wird dem Empfänger die korrekte Reihenfolge für die Bearbeitung angezeigt. Soll eine Signatur vor einer Verschlüsselung erfolgen, so muss zunächst das erstellte <ds:Signature>-Element eingefügt werden. Danach wird das erstellte <xenc:EncryptedData>-Element dem Element <ds:Signature> vorangestellt. Im umgekehrten Fall, wenn eine Signatur nach der Verschlüsselung erstellt werden soll, wird entsprechend erst das <xenc:EncryptedData>-Element eingefügt und dann das <ds:Signature>-Element vorangestellt.
XMLdsig erlaubt es, die Elemente, die signiert werden sollen, anhand unterschiedlicher Methoden zu referenzieren. Dies kann sowohl durch die Angabe einer URI, einer ID oder auch mittels XPath erfolgen. Einige Security Tokens können jedoch evtl. mit diesem Mechanismen nicht referenziert werden. Deshalb wurde in die WSS-Spezifikation das bereits vorgestellte Element <wsse:SecurityTokenReference> (STR) definiert, das es erlaubt alle *Token*-Formate zu referenzieren. Dazu definiert die WSS-Spezifikation zusätzlich einen neuen Transformationsmechanismus, der im <ds:Transform>-Element von XMLdsig verwendet wird (vgl. Kapitel 2.7.1). Der *STR Dereference* Transformationsmechanismus wandelt ein <wsse:SecurityTokenReference>-Element in das von diesem Element referenzierte Security Token um.
Eine Signatur eines STR-Elements ermöglicht es, die Integrität und Authentizität des STR zu gewährleisten, nicht jedoch des referenzierten Security Token (ST). Ist dies erwünscht muss der *STR Dereference Transform (STRDT)*-Mechanismus angewendet werden. Dieser ersetzt für die Berechnung der Signatur die STR durch das ST. Soll beides signiert werden, also das ST und die STR, so muss die Signatur jeweils einmal mit STRDT und einmal ohne STRDT berechnet werden.
In Abbildung 2.42 ist die Anwendung des STRDT (vgl. Zeile 11) dargestellt. Das Beispiel zeigt zudem die Verwendung des <ds:Signature>-Elements im <wsse:Security>-Header und damit innerhalb einer SOAP-Nachricht.
WSS-Security führt des Weiteren das <wsse11:SecurityConfirmation>-Element ein. Es erlaubt, den Wert einer Signatur einer erhaltenen Nachricht wieder in eine Antwortnachricht mit aufzunehmen. Der Inhalt eines <ds:SignatureValue>-Elements wird dazu als Wert des Value-Attributs des <wsse11:SecurityConfirmation>-Elements eingefügt.

```
 1: <S11:Envelope xmlns:S11="..." xmlns:wsse="..."
 2:    xmlns:ds="..." xmlns:wsu="...">
 3:   <S11:Header>
 4:     <wsse:Security>
 5:       <wsse:SecurityTokenReference wsu:Id="MyTokenReference">...
 6:       </wsse:SecurityTokenReference>
 7:       <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
 8:         <ds:SignedInfo>
 9:           <ds:Reference URI="#MyTokenReference">
10:             <ds:Transforms>
11:               <ds:Transform Algorithm="...#STR-Transform">
12:                 <wsse:TransformationParameters>
13:                   <ds:CanonicalizationMethod Algorithm="..."/>
14:                 </wsse:TransformationParameters>
15:               </ds:Transform>
16:             </ds:Transforms>
17:             <ds:DigestMethod Algorithm="...#sha1"/>
18:             <ds:DigestValue>...</ds:DigestValue>
19:           </ds:Reference>
20:         </ds:SignedInfo>
21:         <ds:SignatureValue>...</ds:SignatureValue>
22:       </ds:Signature>
23:     </wsse:Security>
24:   </S11:Header>
25:   <S11:Body wsu:Id="...">...</S11:Body>
26: </S11:Envelope>
```

Abbildung 2.42: WSS-Signatur und STR Dereference Transformation

Dadurch kann der Sender einer Nachricht eindeutig feststellen, dass die erhaltene Antwort als Ergebnis seiner ursprünglichen Anfrage erstellt wurde. Es wird also eine eindeutige Zuordnung von Antwort- zu Anfragenachrichten ermöglicht. Dies kann zur Abwehr von bestimmten Angriffen verwendet werden.

2.8.3.3 Verschlüsselung

Die WSS-Spezifikation erlaubte es, beliebige Kombinationen des SOAP-Body und des SOAP-Header zu verschlüsseln. Hierbei kommt entweder ein symmetrischer Schlüssel zum Einsatz, der den Kommunikationspartnern zuvor bereits bekannt ist oder dieser wird mit asymmetrischen Verschlüsselungsverfahren chiffriert mit der Nachricht transportiert. Die Spezifikation definiert, wie die Elemente `<xenc:ReferenceList>` und `<xenc:EncryptedKey>` der XMLenc-Spezifikation innerhalb des `<wsse:Security>`-Elements angewendet werden können.
Verschlüsselt ein SOAP-Knoten einen Teil einer Nachricht, so muss er die entsprechenden Verschlüsselungselemente in ein bestehendes `<wsse:Security>`-Element an erster Stelle einfügen bzw. eine neues `<wsse:Security>`-Element erstellen und dort die Verschlüsselungselemente einfügen. Die Verschlüsselung der Elemente `<Header>`, `<Envelope>` bzw. `<Body>` ist nicht erlaubt, damit die Struktur eines SOAP-Envelope nach einer Verschlüsselung weiterhin gültig ist.
Mittels des `<xenc:ReferenceList>`-Elements wird im `<wsse:Security>`-Header eine Liste aller chiffrierten Elemente angelegt. Dazu enthält das `<xenc:ReferenceList>`-

Element für jedes <xenc:EncryptedData>-Element des SOAP-Body ein entsprechendes <xenc:DataReference>-Element, das mittels des URI-Attributs auf das verschlüsselte Element verweist.

```
 1: <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
 2:    xmlns:ds="..." xmlns:xenc="...">
 3:   <S11:Header>
 4:     <wsse:Security>
 5:       <xenc:ReferenceList>
 6:         <xenc:DataReference URI="#Enc1"/>
 7:         <xenc:DataReference URI="#Enc2"/>
 8:       </xenc:ReferenceList>
 9:     </wsse:Security>
10:   </S11:Header>
11:   <S11:Body>
12:     <xenc:EncryptedData Id="Enc1">
13:       <ds:KeyInfo><ds:KeyName>Key1</ds:KeyName></ds:KeyInfo>
14:       <xenc:CipherData>
15:         <xenc:CipherValue>...</xenc:CipherValue>
16:       </xenc:CipherData>
17:     </xenc:EncryptedData>
18:     <xenc:EncryptedData Id="Enc2">
19:       <ds:KeyInfo><ds:KeyName>Key2</ds:KeyName></ds:KeyInfo>
20:       <xenc:CipherData>
21:         <xenc:CipherValue>...</xenc:CipherValue>
22:       </xenc:CipherData>
23:     </xenc:EncryptedData>
24:   </S11:Body>
25: </S11:Envelope>
```

Abbildung 2.43: WSS-Verschlüsselung

Die WSS-Spezifikation erlaubt es, dass die im <xenc:ReferenceList>-Element referenzierten <xenc:EncryptedData>-Elemente mit unterschiedlichen Schlüsseln chiffriert werden. Dazu kann für jedes <xenc:EncryptedData>-Element mittels eines Kindelements <ds:KeyInfo> ein eigener Schlüssel angegeben werden. Dies ist ein Gegensatz zur Spezifikation *XMLenc*. Diese definiert das <xenc:ReferenceList>-Element ursprünglich als Kindelement des <xenc:EncryptedKey>-Elements, um dort alle Elemente zu referenzieren, die mit diesem Schlüssel chiffriert wurden.

Abbildung 2.43 zeigt die Verwendung des <xenc:ReferenceList>-Elements. In diesem Beispiel werden zwei verschlüsselte Elemente referenziert, die mit unterschiedlichen Schlüsseln chiffriert sind.

Das <xenc:EncryptedKey>-Element ermöglicht es, einen zuvor chiffrierten Schlüssel im <wsse:Security>-Header abzulegen. Dieser kann selbst eine Liste von referenzierten Elementen enthalten, die mit diesem Schlüssel chiffriert sind. Anderenfalls muss das Element <xenc:EncryptedKey> von einem <xenc:KeyInfo>-Element referenziert werden, das innerhalb der entsprechenden <xenc:EncryptedData>-Elemente eingefügt ist.

Die Spezifikation definiert außerdem noch das Element <wsse11:EncryptedHeader>, das es erlaubt, Header-Elemente einer SOAP-Nachricht zu verschlüsseln. Dabei wird das komplette Header-Element durch das chiffrierte <wsse11:EncryptedHeader>-Element er-

setzt. Ein <wsse11:EncryptedHeader>-Element enthält dazu den verschlüsselten Header in Form eines <xenc:EncryptedData>-Elements. Die ursprünglichen Attribute des Header werden dabei als Attribute in das <wsse11:EncryptedHeader>-Element übernommen. Damit ist gewährleistet, dass die Header von den entsprechenden SOAP-Knoten weiterhin verarbeitet werden können, sofern sie den passenden Schlüssel zur Entschlüsselung besitzen. Die WSS-Spezifikation erlaubt es zudem, dass ein <wsse11:EncryptedHeader>-Element selbst wieder verschlüsselt werden kann. Dies ist vergleichbar mit der *Super-Encryption* der *XMLenc*-Spezifikation (vgl. Kapitel 2.7.2).

2.8.3.4 Fehlerbehandlung innerhalb von WS-Security

Bei der Verwendung von WS-Security können eine Reihe von Fehlern auftreten. Beispielsweise kann es vorkommen, dass ein verwendetes Security Token oder ein kryptografischer Algorithmus vom Empfänger nicht unterstützt wird. In diesem Fall kann der Empfänger dem Absender dies mit einem *SOAP-Fault* mitteilen (vgl. Kapitel 2.4.3.2).
In Tabelle 2.9 sind die SOAP-Fehlercodes aufgelistet, die von der WSS-Spezifikation definiert werden. Die Tabelle enthält darüber hinaus den zugehörigen *Fault-String*.

Fault-Code	Fault-String
wsse:UnsupportedSecurityToken	An unsupported token was provided
wsse:UnsupportedAlgorithm	An unsupported signature or encryption algorithm was used
wsse:InvalidSecurityToken	An invalid security token was provided
wsse:FailedAuthentication	The security token could not be authenticated
wsse:FailedCheck	The signature or decryption was invalid
wsse:SecurityTokenUnavailable	A referenced security token could not be retrieved
wsse:MessageExpired	The message has expired

Tabelle 2.9: WS-Security SOAP-Faults

2.8.4 WS-Policy

Die *WS-Policy*-Spezifikation [50] liegt derzeit in der Version 1.2 vor und ist im April 2006 beim W3C eingereicht worden. Sie ermöglicht es, detaillierte Anforderungen für Web Services zu spezifizieren. Dazu definiert die Spezifikation eine flexible und erweiterbare Grammatik, um allgemeine Präferenzen, Anforderungen und Fähigkeiten von Instanzen innerhalb einer auf Web Service basierenden Infrastruktur zu formulieren. Diese Eigenschaften werden in Form von so genannten *Policies* (Richtlinien) ausgedrückt. Eine *Policy* wird als eine Sammlung von *Policy-Alternatives* definiert, wobei jede dieser Alternativen eine Menge von so genannten *Policy-Assertions* (Zusicherungen) bildet.
Jede *Assertion* spezifiziert dabei eine bestimmte Präferenz, Anforderung oder Fähigkeit. Diese können beispielsweise festlegen, welcher Authentifikationmechanismus, welches Transport-Protokoll oder auch welche Dienstgüte (*Quality of Service (QoS)*) gefordert wird. WS-Policy definiert nur die Grammatik für die Spezifikation solcher *Assertions*. Welche Anforderungen damit ausgedrückt werden, ist meist anwendungsabhängig und erfordert daher eine eigene Spezifikation. Für die Definition von Sicherheitsanforderung existiert bereits die *WS-SecurityPolicy*-Spezifikation, die in Kapitel 2.8.11 vorgestellt wird.
Das WS-Policy-Schema [50] spezifiziert eine Policy durch das Element <wsp:Policy>. Die-

ses Element kann die Kindelemente <wsp:PolicyReference>, <wsp:ExactlyOne> oder <wsp:All> aufnehmen und außerdem selbst auch wieder aus einem <wsp:Policy>-Element bestehen. Die Kindelemente einer WS-Policy werden auch als *Policy-Operatoren* bezeichnet. Innerhalb der Operatoren sind als Kindelemente die einzelnen *Assertions* definiert. *Assertions* können mit dem Attribut wsp:Optional als optional markiert werden. Zudem kann ein Policy-Operator selbst wieder andere Policy-Operatoren enthalten.

```
 1: <wsp:Policy xmlns:wsp="..." ...>
 2:   <wsp:All>
 3:     <wsp:Policy>
 4:       <wsp:PolicyReference URI="#EineAssertion"/>
 5:       <Assertion ...> ... </Assertion>
 6:       <Assertion ...> ... </Assertion>
 7:     </wsp:Policy>
 8:     <wsp:Policy>
 9:       <wsp:ExactlyOne>
10:         <Assertion ...> ... </Assertion>
11:         <Assertion ...> ... </Assertion>
12:       </wsp:ExactlyOne>
13:     </wsp:Policy>
14:   </wsp:All>
15: </wsp:Policy>
```

Abbildung 2.44: Beispielstruktur einer WS-Policy

Wird das <wsp:Policy>-Element als Operator verwendet, so ist dies mit dem Operatorelement <wsp:All> gleichzusetzen. Das <wsp:All>-Element fordert, dass alle eingeschlossenen *Assertions* erfüllt werden müssen. Mit dem <wsp:ExactlyOne>-Element wird ausgedrückt, dass genau eine der eingeschlossenen *Assertions* erfüllt werden muss. Das Element <wsp:PolicyReference> ermöglicht es, *Assertions* wiederzuverwenden, die an einer anderen Stelle definiert sind. Semantisch bedeutet die Verwendung einer solchen Referenz, dass das Element <wsp:PolicyReference> durch ein <wsp:All>-Element ersetzt wird und der Inhalt der referenzierten Policy als Kindelement in dieses Element eingefügt wird.
Durch diese Struktur können sehr komplexe Policies aufgebaut werden. Abbildung 2.44 zeigt eine Policy, die sich aus zwei Policies zusammensetzt. Beide Policies müssen erfüllt werden, wobei sich erstere aus einer Referenz auf eine Assersion sowie zwei direkt angegebenen Assertions zusammensetzt. Die zweite Policy definiert zwei Assertions, wobei jeweils eine erfüllt werden muss.
Die WS-Policy-Spezifikation definiert nicht wie Policies abgerufen oder wie sie an einen Web Service gebunden werden können. Dies ist jeweils abhängig von der verwendeten Technologie und muss daher separat spezifiziert werden. Für die Verknüpfung von Policies mit beliebigen XML-Elementen, Teilen einer WSDL-Beschreibung oder UDDI-Elementen existiert die *WS-PolicyAttachment*-Spezifikation, die in Kapitel 2.8.10 vorgestellt wird.

2.8.5 WS-Trust

Die *WS-Trust*-Spezifikation liegt derzeit in der Version 1.3 [68] als Entwurf beim *OASIS Web Services Secure Exchange Technical Commitee (OASIS WS-SX TC)* vor. Die Spezifikation beschreibt, wie Vertrauensbeziehungen in der Web Service-Welt aufgebaut werden können.

Die Spezifikation nutzt dazu die Mechanismen der WS-Security-Spezifikation und definiert zusätzliche Funktionen und Erweiterungen für den Austausch, die Ausgabe und die Verteilung von Security Tokens zwischen verschiedenen Vertrauensdomänen.

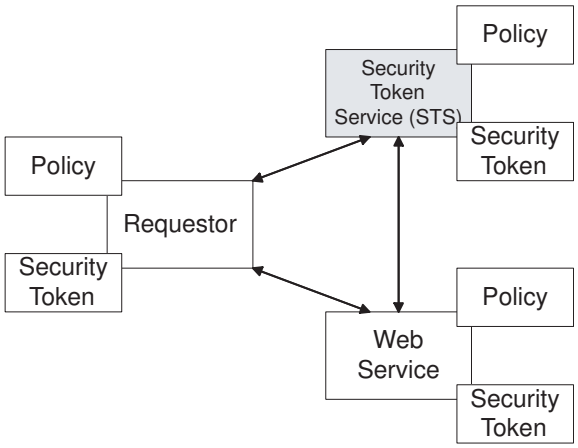

Abbildung 2.45: WS-Trust-Modell

Um eine sichere Kommunikation zwischen zwei Partnern zu ermöglichen, müssen diese entsprechende Sicherheitszeugnisse (*Credentials*) entweder direkt oder indirekt austauschen. Dabei muss sichergestellt werden, dass die jeweiligen Kommunikationspartner den Credentials des anderen Kommunikationspartners vertrauen können. Die WS-Trust-Spezifikation definiert dafür Methoden zur Ausgabe, Erneuerung, Prüfung und Entwertung von Security Tokens sowie zur Bewertung und Vermittlung von Vertrauensbeziehungen.[68]
Das WS-Trust-Modell ist in Abbildung 2.45 dargestellt. Es zeigt das Zusammenspiel von einem Web Service-Anbieter, einem Nutzer und dem *Security Token Service (STS)*. Alle beteiligten Instanzen können selbst als Web Services implementiert sein und besitzen daher ggf. eigene Policies und erlauben den Zugriff über entsprechende Security Tokens. Die WS-Policy-Spezifikation findet hier Verwendung, um die einzelnen Policies zu formulieren. Die Pfeile in der Abbildung repräsentieren mögliche Kommunikationswege. So könnte eine Web Service-Nutzer ein Security Token vom STS beziehen und damit die autorisierte Nutzung dieses Tokens für den Web Service demonstrieren. Der Web Service kann dem erhaltenen Token des Nutzers entweder direkt Vertrauen, wenn er eine Vertrauensbeziehung zum STS besitzt oder eine Validierung des Tokens bei seinem vertrauenswürdigen STS veranlassen.

```
1: <wst:RequestSecurityToken Context="..." xmlns:wst="...">
2:   <wst:TokenType>...</wst:TokenType>
3:   <wst:RequestType>...</wst:RequestType>
4:   ...
5: </wst:RequestSecurityToken>
```

Abbildung 2.46: Allgemeiner Aufbau des RequestSecurityToken

Der STS ist der zentrale Dienst des WS-Trust-Modells, der die Ausgabe, Erneuerung, Prüfung und Entwertung von Security Tokens übernimmt. Hierfür werden in der Spezifikation jeweils so genannte *Bindings* definiert, die entsprechende Elemente und Parameter für die einzelnen Protokollnachrichten festlegen. Alle Bindings verwenden ein so genanntes *STS Framework*, das

sich auf zwei Nachrichtentypen beschränkt. Diese Typen werden durch zwei XML-Elemente repräsentiert:
Das `<wst:RequestSecurityToken>`-Element (RST), dessen allgemeiner Aufbau in Abbildung 2.46 dargestellt ist, dient dazu ein Security Token für verschiedene Nutzungszwecke von einem STS abzurufen. Das obligatorische Element `<wst:RequestType>` legt mittels einer URI (`http://docs.oasis-open.org/ws-sx/ws-trust/200512/...`) fest, welches Binding bei der Anfrage verwendet wird und damit die Aktion, die vom STS ausgeführt wird:

- `http://.../200512/Issue`: Ausstellen eines Token
- `http://.../200512/Renew`: Erneuern eines Token
- `http://.../200512/Cancel`: Entwerten eines Token beim STS
- `http://.../200512/STSCancel`: Entwerten eines Token beim Nutzer
- `http://.../200512/Validate`: Prüfen eines Token

Das optionale Element `<wst:TokenType>` legt mittels einer URI fest, welcher Token-Typ vom STS ausgestellt werden soll. Diese URIs sind in den entsprechenden Token-Profilen der WS-Security-Spezifikation (vgl. Kapitel 2.8.3) definiert.
Das `<wst:RequestSecurityTokenResponse>`-Element (RSTR) wird als Antwort auf eine RST-Anfrage gesendet und enthält entweder das angefragte Security Token oder das Ergebnis der ausgewählten Aktion bzw. des Bindings. In Abbildung 2.47 ist der generelle Aufbau des RSTR-Elements dargestellt. Auch hier kann mit dem Element `<wst:TokenType>`der Typ des Token festgelegt werden. Das `<wst:RequestedSecurityToken>`-Element enthält dann das ausgestellte Token oder referenziert es.
Die einzelnen Bindings legen schließlich noch spezifische Kindelemente für diese zwei Nachrichtentypen fest. Eine Auflistung und genaue Erklärung befindet sich in den Kapiteln 4-7 der WS-Trust-Spezifikation [68].

```
1: <wst:RequestSecurityTokenResponse Context="..." xmlns:wst="...">
2:   <wst:TokenType>...</wst:TokenType>
3:   <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
4:   ...
5: </wst:RequestSecurityTokenResponse>
```

Abbildung 2.47: Allgemeiner Aufbau des RequestSecurityTokenResponse

2.8.6 WS-Privacy

WS-Privacy ist eine Spezifikation, die in der genannten WS-Security Roadmap [57] vorgeschlagen wurde. Eine Veröffentlichung lag zum Zeitpunkt der Erstellung dieser Arbeit jedoch noch nicht vor. Dennoch soll hier ein grober Überblick über die Idee hinter dieser Spezifikation gegeben werden.
Der Datenschutz (*Privacy*) wird durch die Einhaltung der Informationsvertraulichkeit und zusätzlich durch eine entsprechende Zugriffskontrolle gewährleistet. *WS-Privacy* schlägt dazu eine Kombination der Spezifikationen WS-Policy, WS-Security und WS-Trust vor, die es erlauben

soll, so genannte *Privacy Policies* (Datenschutzrichtlinien) zu formulieren. Diese Privacy Policies werden von Unternehmen oder Organisationen formuliert, die Web Services anbieten. Jeder Zugriff auf diese Web Services erfordert nun, dass der Anfragende die Einhaltung der Datenschutzrichtlinien bestätigt. Dies erfolgt durch Angabe von speziellen Versicherungen bzw. Behauptungen (*Claims*) innerhalb einer SOAP-Nachricht. Diese Claims werden mittels WS-Security in entsprechende Security Tokens eingebettet.
Das heißt, dass mit WS-Privacy definiert wird, wie Anforderungen an den Datenschutz in eine WS-Policy-Beschreibung eingebettet werden können. WS-Trust definiert dann, wie die in einer SOAP-Nachricht mit Hilfe von WS-Security eingebetteten Datenschutzrichtlinien ausgewertet werden können.

2.8.7 WS-SecureConversation

Die *WS-SecureConversation*-Spezifikation [47] wurde im Jahr 2005 unter Zusammenarbeit von IBM, BEA Systems, Microsoft und vielen weiteren Unternehmen veröffentlicht. Derzeit liegt die Version 1.3 als Entwurf beim *OASIS Web Services Secure Exchange Technical Commitee (OASIS WS-SX TC)*) vor.
Die in der WS-Security-Spezifikation definierten Mechanismen erlauben es, einzelne Nachrichten hinsichtlich verschiedener Sicherheitsaspekte abzusichern. Werden jedoch mehrere Nachrichten an einen Adressaten gesendet, so muss jede dieser Nachrichten einzeln abgesichert werden. Das bedeutet bei jedem Kommunikationsvorgang müssen entsprechende Signaturen überprüft werden, Security Tokens angefordert werden, die Einhaltung von Policies sichergestellt werden und neue temporäre symmetrische Schlüssel mit dem öffentlichen Schlüssel des Empfängers verschlüsselt werden.
Die WS-SecureConversation-Spezifikation definiert nun Mechanismen, die diesen sehr rechenintensiven Prozess nur einmal zu Beginn einer Konversation notwendig machen. Ähnlich zu vergleichbaren Sicherheitsprotokollen, wie beispielsweise SSL auf der Transportebene, baut der SecureConversation-Mechanismus einen gemeinsamen Sicherheitskontext auf Nachrichtenebene auf.
Im Wesentlichen werden für den Aufbau des Sicherheitskontexts die Mechanismen der WS-Trust-Spezifikation wiederverwendet. Dies bedeutet, dass eine der Parteien (der Service-Nutzer) zunächst eine RST-Nachricht an ein STS sendet. Diese antwortet wie gewöhnlich mit einer RSTR-Nachricht, die ein so genanntes *Security Context Token (SCT)* enthält bzw. eine Referenz auf dieses beinhaltet. Das SCT besteht aus einem systemweit eindeutigen Identifikator für den gemeinsamen Sicherheitskontext. Zudem wird mit dem `<wst:RequestedProofToken>`-Element auf den geheimen symmetrischen Schlüssel für den Sicherheitskontext verwiesen. Dieser ist jeweils mit dem öffentlichen Schlüssel der Dialogpartner chiffriert in diesem Element abgelegt.
Ohne selbst eine RST-Nachricht an den STS gesendet zu haben, erhält der Kommunikationspartner ebenfalls eine RSTR-Nachricht. Dieser Vorgang wird als *Token Propagation* bezeichnet und ermöglicht erst den Aufbau des gemeinsamen Sicherheitskontexts. Nun kennen beide Kommunikationspartner das SCT und können nun die Kommunikation mit den gewohnten WS-Security-Mechanismen aufbauen, wobei sie für Signaturen und Verschlüsselung den Sicherheitskontext referenzieren. Abbildung 2.48 verdeutlicht diesen Ablauf noch einmal.
Der gemeinsame Sicherheitskontext kann auch ohne ein STS aufgebaut werden. Dabei erstellt eine Partei das entsprechende SCT und sendet dieses mit den in der WS-Trust-Spezifikation definierten Mechanismen, an die andere Partei. Dieses Szenario ist natürlich nur dann sinnvoll, wenn sich die einzelnen Parteien bereits kennen und ein Vertrauensverhältnis besteht.

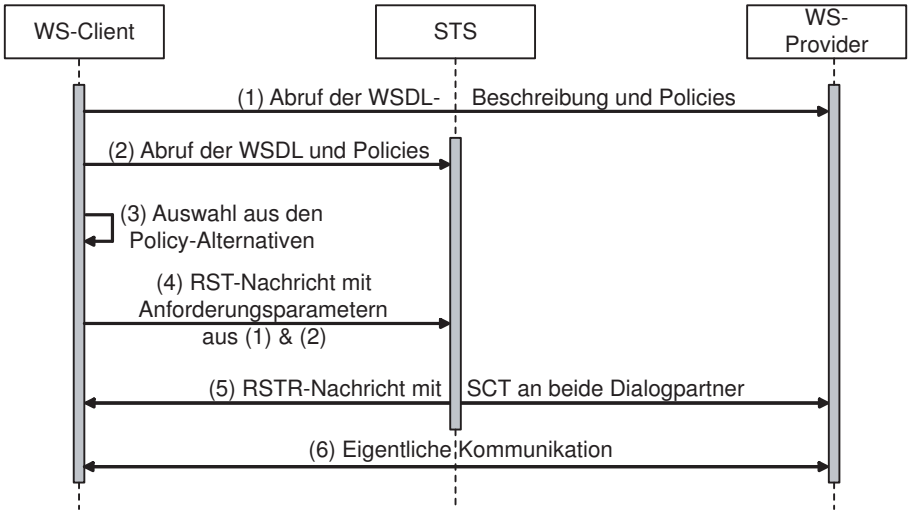

Abbildung 2.48: Aufbau einer WS-SecureConversation

In Abbildung 2.49 ist der generellen Aufbau des SCT dargestellt. Das obligatorische Element `<wsc:Identifier>` enthält den eindeutigen Identifikator des SCT. Die Spezifikation erlaubt es, den Sicherheitskontext zu erneuern bzw. neue Schlüssel für diesen zu vergeben. Damit eine so erstellte neue Version eines SCT identifiziert werden kann, wird das `<wsc:Instance>`-Element verwendet.

```
1: <wsc:SecurityContextToken wsu:Id="..." ...>
2:   <wsc:Identifier>...</wsc:Identifier>
3:   <wsc:Instance>...</wsc:Instance>
4:   ...
5: </wsc:SecurityContextToken>
```

Abbildung 2.49: Allgemeiner Aufbau des SecurityContextToken

Die Spezifikation empfiehlt, als Prophylaxe gegen Kryptoanalyse und mögliche Rückschlüsse auf die geheimen Schlüsseldaten, regelmäßig neue Schlüssel aus dem Basisschlüssel abzuleiten. Hierbei wird der Binärwert des Basisschlüssels, der Bezeichner des Kommunikationspartners und eine Nonce verwendet, um mit Hilfe einer kryptografischen Hash-Funktion einen neuen Schlüssel zu berechnen. Die Kommunikationspartner verweisen dann statt auf das SCT auf das Element `<wsc:DerivedKeyToken>`, das dem Empfänger einer Nachricht die notwendigen Parameter liefert, um seinerseits den neuen Schlüssel zu berechnen.

Die Spezifikation definiert des Weiteren, wie die in einem SCT beinhalteten Versicherungen oder Behauptungen (*Claims*) berichtigt werden können. Dazu existiert das *SCT Amend binding*. Die Gültigkeit eines SCT ist in den meisten Fällen zeitlich begrenzt. Die Spezifikation erlaubt es jedoch, durch die Wiederverwendung des *Renewal*-Binding der WS-Trust-Spezifikation, die Laufzeit zu verlängern. Um ein SCT zu entwerten, kann das *Cancel*-Binding der WS-Trust-Spezifikation verwendet werden.

2.8.8 WS-Federation

Die Version 1.1 der *WS-Federation-Language (WS-Federation)*-Spezifikation [102] wurde im Jahr 2006 veröffentlicht. Sie definiert Mechanismen, mit denen sich unterschiedliche Sicherheitsdomänen zu einem Verbund vereinigen lassen, um eine gemeinsame Vertrauensgemeinschaft (*Trust Federation*) zu bilden. Die Spezifikation beschreibt, wie dies unter Anwendung von WS-Security, WS-Policy, WS-Trust und WS-SecureConversation ermöglicht wird.

Unter zu Hilfenahme von WS-Federation wird es möglich, dass sich ein Benutzer, durch Verwendung eines X.509 Zertifikats, bei einer Sicherheitsdomäne authentifiziert und danach einen Web Service einer anderen Sicherheitsdomäne nutzt, die eigentlich nur eine Authentifikation mittels Kerberos-Tickets unterstützt. Die Spezifikation definiert Mechanismen für ein SSO-Szenario (siehe Kapitel 2.5) in einer Infrastruktur, deren Systeme unterschiedliche Sicherheitstechnologien einsetzen.

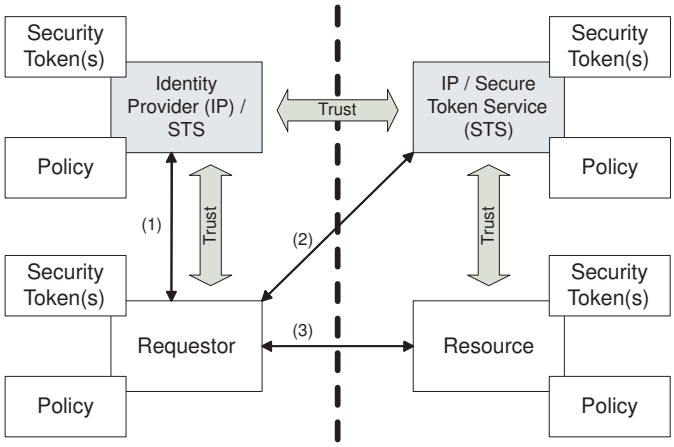

Abbildung 2.50: Topologie eines Federation Modells

Abbildung 2.50 zeigt eine mögliche Topologie eines WS-Federation-Modells. Da beide Sicherheitsinfrastrukturen auf Basis des Vertrauensverhältnisses zwischen den *Identiy Providern* bzw. *Secure Token Services* eine so genannte *Federation* bilden, können die Nutzer der einen Sicherheitsinfrastruktur die Web Services der anderen Sicherheitsinfrastruktur nutzen.

Ein Web Service-Nutzer ruft (1) in diesem Beispiel zunächst ein Security Token von seinem STS ab. In Schritt (2) verwendet er dieses, um ein Security Token vom STS der anderen Sicherheitsinfrastruktur abzurufen. Danach kann er das neue Security Token verwenden, um sich gegenüber dem Service zu authentifizieren (3). Hier wird also das Token des einen STS gegen ein Token des anderen STS ausgetauscht bzw. von diesem nach Überprüfung neu signiert, so dass der Service das Token für den Zugriff dieses Benutzers akzeptiert.

Die in dieser Spezifikation definierten Mechanismen können sowohl von passiven als auch aktiven *Requestors* (Anforderer einer Web Service-Funktion) benutzt werden. Ein aktiver Requestor verwendet die WS*-Mechanismen und kann somit auf gewohnte Weise mit einem Web Service-Anbieter interagieren. Das *WS-Federation: Active Requestor Profile* [54] beschreibt die Mechanismen für diese Art von Requestors. Ein passiver Requestor ist beispielsweise ein Webbrowser. Er ist nicht in der Lage, SOAP und damit die WS*-Nachrichten zu verstehen. Das *WS-Federation: Passive Requestor Profile* [55] beschreibt für diesen Typ entsprechende HTTP-Nachrichten, die es einem Webbrowser erlauben, mit einem STS zu kommunizieren.

2.8.9 WS-Authorization

Die *WS-Authorization*-Spezifikation soll definieren, wie Zugriffsrichtlinien für Web Services spezifiziert und verwaltet werden können. Im Detail soll sie spezifizieren, wie entsprechende Versicherungen bzw. Behauptungen (*Claims*) in ein Security Token eingebettet und wie diese am Endpunkt interpretiert werden können.[57]

Anhand der Spezifikation soll ein flexibles und erweiterbares System hinsichtlich des Autorisierungsformats und der Autorisierungssprache entwickelt werden [57].

Da zum Zeitpunkt der Erstellung dieser Arbeit die WS-Authorization-Spezifikation noch nicht veröffentlicht war, können hier keine weiteren Angaben gemacht werden. Es sei jedoch daraufhin gewiesen, dass diese Spezifikation wahrscheinlich Überschneidungen mit der XACML (vgl. Kapitel 2.8.12) besitzen wird.

2.8.10 WS-PolicyAttachement

Die *WS-PolicyAttachment*-Spezifikation Version 1.2 [49] wurde im März 2006 veröffentlicht und im April 2006 beim W3C eingereicht. Sie definiert verschiedene Mechanismen um Policies an Subjekte zu binden. Eine Policy alleine betrachtet ist nicht in der Lage, das Subjekt zu beschreiben, für das sie gelten soll. Diese Trennung erlaubt es, Policies wiederzuverwenden und sie mehreren verschiedenen Subjekten zuzuordnen. Als Subjekt kann beispielsweise ein beliebiges XML-Dokument, eine WSDL-Nachricht oder ein Web Service-Endpunkt verstanden werden.

Die WS-PolicyAttachment-Spezifikation definiert zwei grundlegende Mechanismen Policies mit Subjekten zu verbinden:

XML-Element Attachments Hier werden die *Policy Attachments* direkt in die XML Elemente eingebettet, die die Subjekte beschreiben. Die Zuordnung von Policies zu einem XML-Element erfolgt hier entweder durch ein Attribut oder durch eine Kindelement, das die jeweilige URI einer anzuwendenden Policy referenziert.

Externe Policy Attachments Die Zuordnung der Policy zu einem Subjekt erfolgt hier außerhalb der eigentlichen Definition des Subjekts. Dazu definiert die Spezifikation das Element `<wsp:PolicyAttachment>`, das eine oder mehrere Policies enthalten oder referenzieren kann und diese mit einem oder mehreren Subjekten verbindet. Diese Verbindung wird über eine so genannte *Domain Expression* hergestellt. Ein Beispiel für eine solche *Domain Expression* ist ein Web Service-Endpunkt. Optional kann das `<wsp:PolicyAttachment>`-Element verschiedene Sicherheitsinformationen aufnehmen, in dem es ein `<wsse:Security>`-Tag der WS-Security-Spezifikation als Kindelement zulässt.

Die Spezifikation definiert außerdem, wie Mechanismen wie Policies mit WSDL und UDDI verknüpft werden können. Dabei wird festgelegt, wie Policies von einer WSDL-Definition referenziert werden und wie Policies mit einem bereits veröffentlichen Web Service-Endpunkt verknüpft werden können. Zudem wird spezifiziert, wie Policies mit UDDI-Entitäten verbunden werden können. Eine genaue Erklärung der einzelnen Mechanismen würde an dieser Stelle zu weit führen; es sei daher auf die Kapitel 4 und 5 der WS-PolicyAttachment-Spezifikation verwiesen [49].

2.8.11 WS-SecurityPolicy

Die *WS-SecurityPolicy*-Spezifikation Version 1.1 wurde im Jahr 2005 von IBM, Microsoft, RSA Security und VeriSign veröffentlicht. Derzeit liegt ein Entwurf der Version 1.2 [56] beim *OASIS Web Services Secure Exchange Technical Commitee (OASIS WS-SX TC)*) vor. Die Spezifikation liefert ein umfangreiches Vokabular zur Beschreibung von Sicherheitsanforderungen an einen Web Service. Diese werden in Form von der in der WS-Policy-Spezifikation (vgl. Kapitel 2.8.4) definierten Policies ausgedrückt. WS-SecurityPolicy ermöglicht es, *WS-Policy Assertions* zu spezifizieren, die es erlauben, mögliche Sicherheitsanforderungen an einen Web Service zu formulieren.
Die Spezifikation definiert zunächst so genannte *Protection Assertions*. Sie beinhalten *Integrity-* und *Confidentiality-Assertions* und ermöglichen es detailliert festzulegen, welche Teile einer SOAP-Nachricht signiert bzw. verschlüsselt werden müssen. Die einzelnen Elemente oder Teile eines XML-Dokuments können einerseits durch Angabe von so genannten *QNames* [120] oder andererseits durch XPath-Ausdrücke [106] identifiziert werden. Die *Required Elements Assertions* erlauben es zudem festzulegen, welche Elemente grundsätzlich in einem SOAP-Header vorhanden sein müssen. Diese Elemente können wiederum durch XPath-Ausdrücke festgelegt werden.
In Abbildung 2.51 sind drei Beispiele von *Protection Assertions* dargestellt. Zeile 1-3 zeigt eine *Assertion*, die eine Signatur eines SOAP-Header unter Angabe eines QName fordert. In Zeile 4-6 wird die Verschlüsselung eines Teils einer SOAP-Nachricht gefordert, der anhand eines XPath-Ausdrucks bestimmt wird. Die Zeilen 7-9 erfordern, dass ein bestimmtes Element im SOAP-Header vorhanden sein muss.

```
1: <sp:SignedParts ... >
2:   <sp:Header Name="Header1"/>
3: </sp:SignedParts>

4: <sp:EncryptedElements XPathVersion="...">
5:   <sp:XPath>XPath-Ausdruck</sp:XPath>
6: </sp:EncryptedElements>

7: <sp:RequiredElements XPathVersion="...">
8:   <sp:XPath>XPath-Ausdruck</sp:XPath>
9: </sp:RequiredElements>
```

Abbildung 2.51: WS-SecurityPolicy - Protection Assertions

Mittels so genannten *Token Assertions* können Anforderungen an Security Tokens spezifiziert werden. Es kann festgelegt werden, ob ein *Token* innerhalb einer Nachricht enthalten sein muss oder ob es auch außerhalb referenziert werden darf. Zudem kann der geforderte Typ eines *Token* definiert werden. Die Spezifikation enthält dazu entsprechende Elemente mit denen alle gängigen Typen von *Tokens* bestimmt werden können. Hierfür existieren beispielsweise die Elemente `<sp:UsernameToken>`, `<sp:X509Token>` und `<sp:KerberosToken>`. Für jeden *Token*-Typ können jeweils spezielle Anforderungen und Eigenschaften (z.B die geforderte Version) angegeben werden. Die speziellen Elemente `<sp:SecurityContextToken>` und `<sp:SecureConversationToken>` erlauben es, *Assertions*, für die in der Spezifikation *WS-SecureConversation* (vgl. Kapitel 2.8.7) definierten *Tokens*, zu formulieren.
Die so genannten *Security Binding Assertions* ermöglichen es, weitere spezielle *Assertions* zu

formulieren, die die Sicherung auf Transport- oder Nachrichtenebene betreffen. So kann mit der *AlgorithmSuite Assertion* festgelegt werden, welche kryptografischen Algorithmen verwendet werden sollen. Die WS-SecurityPolicy-Spezifikation definiert dazu eine Menge von Bezeichnern, die jeweils für die unterschiedlichen Anwendungsbereiche (Verschlüsselung, Signatur, Hash-Wert, Schlüsselableitung, ...) entsprechende kryptografischen Algorithmen festlegen und deren Parameter spezifizieren. In der *AlgorithmSuite Assertion* erfolgt dann nur noch ein Verweis auf solch einen Bezeichner.
Zudem können mit den *SymmetricBinding-* und *AsymmetricBinding Assertions* spezielle Anforderungen an die symmetrischen und asymmetrischen Kryptotechniken formuliert werden.

2.8.12 Extensible Access Control Markup Language

Die *Extensible Access Control Markup Language (XACML)* [104] ist ein OASIS-Standard, der es Unternehmen ermöglichen soll, Regeln für die Autorisierung von Benutzeraktionen in standardisierter auf XML basierender Form zu formulieren. Die Regeln spezifizieren welche Zugriffsrechte eine Person, ein Web Service oder ein System besitzt bzw. nicht besitzt.
Mit XACML ist es möglich, die Zugriffsverwaltung an einem zentralen Punkt vorzunehmen. Dies vereinfacht die Verwaltung, spart Kosten und verringert das Sicherheitsrisiko durch fehlerhafte Konfigurationen.
XACML setzt für die sichere Anwendung eine vorhergehende Authentifikation voraus und wird daher in den meisten Fällen in Kombination mit SAML eingesetzt. Nach einer erfolgreichen Authentifikation wird der Zugang zu einer Ressource anhand der spezifizierten XACML-Regel entschieden.
Die Regeln werden durch das Element `<Rule>` spezifiziert. Innerhalb dieses Elements wird festgelegt, welchen Benutzern (`<Subjects>`) unter welchen Bedingungen Zugriff auf bestimmte Ressourcen (`<Resources>`) mittels bestimmter Aktionen (`<Actions>`) gewährt oder verweigert wird. Zudem können Regeln mittels des Elements `<Conditions>` weiteren Restriktionen unterzogen werden. So kann beispielsweise festgelegt werden, dass die Regel nur in einer bestimmten Zeitspanne gültig ist.
Die XACML-Spezifikation erlaubt es, eine Menge von Regeln in einem `<Policy>`-Element zusammenzufassen. Durch die Angabe eines Algorithmus wird festgelegt, wie die Regeln kombiniert werden müssen und damit wie sie ausgewertet werden. Durch das Element `<Target>` können wiederum die Benutzer, Ressourcen und Aktionen der Policy festgelegt werden.
Da XACML ebenfalls auf XML basiert, bietet es eine ideale Möglichkeit für die Umsetzung der Autorisierung in der Web Service-Welt.

2.8.13 XML Key Management Specification

Die *XML Key Management Specification (XKMS)* [109] ist eine Spezifikation des W3C. Es spezifiziert Web Services, die die Verteilung und Registrierung sowie das Management öffentlicher Schlüssel ermöglichen. Die Spezifikation dient damit zum Austausch von Schlüsselinformationen, die für die Verwendung von XMLdsig und XMLenc von grundlegender Bedeutung sind und somit eine wichtige Rolle für die Anwendung des WS-Security-Standards spielt.
Das Hauptziel der XKMS-Spezifikation ist es, die Komplexität einer benötigten PKI-Infrastruktur für den Entwickler einer sicheren Anwendung zu verstecken. Der Entwickler einer Anwendung soll somit eine PKI-Infrastruktur nutzen können, ohne sich im Detail mit dieser Struktur befassen zu müssen. XKMS transferiert zu diesem Zweck diese Komplexität von der Anwendung zu einem Service, der das Management der PKI übernimmt. Die Anwendung greift nun

über die Schnittstelle dieses Dienstes auf die Funktionen der PKI zu.
XKMS definiert für diesen Zweck zwei grundlegende Schnittstellen in Form von Web Services. Die *XML Key Information Service Specification (X-KISS)* beschreibt, wie öffentliche Schlüssel lokalisiert und validiert werden können. Für das Lebenszyklusmanagement von Schlüsseln existiert die *XML Key Registration Service Specification (X-KRSS)*.
Grundlegendes Element für die XKMS-PKI ist das so genannte *Key Binding*. Es legt die Zuordnung eines öffentlichen Schlüssels zu einem privaten Schlüssel fest und assoziiert diese mit einer Identität. Des Weiteren spezifiziert das *Key Binding* den Gültigkeitszeitraum sowie die kryptografischen Funktionen (Signatur, Verschlüsselung, ...) für die der Schlüssel verwendet werden darf. Zusätzlich kann auch festgelegt werden, mit welchen Protokollen (SSL, IPSec, ...) der Schlüssel verwendet werden kann.
Das *Key Binding* wird durch ein `<Key Binding>`-Element repräsentiert. Die Informationen über den öffentlichen Schlüssel in diesem Element werden durch Wiederverwendung des Elements `<ds:KeyInfo>` der XMLdsig-Spezifikation abgelegt.
X-KISS stellt zwei Services zur Verfügung. Der `Locate`-Service dient dazu, den Schlüssel eines Benutzers zu finden. Hiermit kann beispielsweise der öffentliche Schlüssel eines Benutzers gesucht werden, der für die Verwendung zur Verschlüsselung oder Signatur von E-Mail-Nachrichten verwendet werden kann. Über den `Validate`-Service kann die Gültigkeit eines Schlüssels überprüft werden.
X-KRSS stellt vier Dienste bereit. Der `Register`-Service erlaubt es, sich bei einer PKI zu registrieren. Mit dem `Recover`-Service wird ein spezieller Dienst bereitgestellt, mit dem bei Verlust des privaten Schlüssels dieser wieder hergestellt werden kann. Der `Reissue`-Service erlaubt es, die Gültigkeitsdauer eines Schlüssels zu verlängern. Für die Entwertung von Schlüsseln existiert noch der `Revoke`-Service.
Die XKMS-Spezifikation ist zwar für die sichere Kommunikation mit Web Services nicht zwingend erforderlich, vereinfacht jedoch die Implementierung sicherer Web Services erheblich.

2.8.14 Weitere WS*-Spezifikationen

Die WS*-Spezifikationen umfassen weitere Standards und Spezifikationen. Im Folgenden werden einige dieser Spezifikationen kurz vorgestellt.

- **WS-Addressing** Die *WS-Addressing*-Spezifikation ist für viele andere WS*-Spezifikationen unverzichtbar. Sie wurde ursprünglich von IBM, BEA Systems, Microsoft, SAP AG und Sun Microsystems entwickelt und liegt derzeit in der Version 1.0 [121] beim W3C als Empfehlung vor. Mittels WS-Addressing können Referenzen auf Web Service-Endpunkte ausgedrückt und bearbeitet werden. Die so genannte *Endpoint Reference* ist eine Datenstruktur, die alle benötigten Daten kapselt, um einen Service Endpunkt zur Laufzeit zu erreichen. Die *Message Information Header* erlauben es, einen Kontext (synchron, asynchron, one-way, ...) für den Nachrichtenaustausch herzustellen. So kann festgelegt werden, an wen eine Nachricht gesendet (`ReplyTo`) werden soll oder wer bei einem Fehler benachrichtigt (`FaultTo`) werden muss.

- **WS-MetaDataExchange** Metadaten in WSDL-Dokumenten, XML-Schemata oder WS-Policies können notwendig sein, um einen Web Service erfolgreich ansprechen zu können. Die *WS-MetaDataExchange*-Spezifikation Version 1.1 [48] beschreibt, wie diese Informationen abgerufen werden können. Die Spezifikation definiert dazu beispielsweise eine Möglichkeit mittels einer *HTTP-Get* Anfrage.

- **WS-Transfer** Die *WS-Transfer*-Spezifikation [103] beschreibt ein auf SOAP basierendes Protokoll zum Zugriff auf XML-Repräsentationen von auf Web Service basierenden Ressourcen.

- **WS-ReliableMessaging** Die *WS-ReliableMessaging*-Protokollspezifikation [53] definiert einen Kontrollmechanismus, mit dessen Hilfe Sequenzen von SOAP-Nachrichten zuverlässig übertragen werden können. Der Empfänger bestätigt dazu den Erhalt der Nachrichten beim Sender und fragt verlorene Nachrichten erneut an. Dies wird durch so genannte *Reliable Messaging (RM)* Komponenten realisiert, die jeweils auf Sender- und Empfängerseite die zuverlässige Nachrichtenübertragung gewährleisten. In Abbildung 2.52 ist dieser Sachverhalt im Überblick dargestellt.

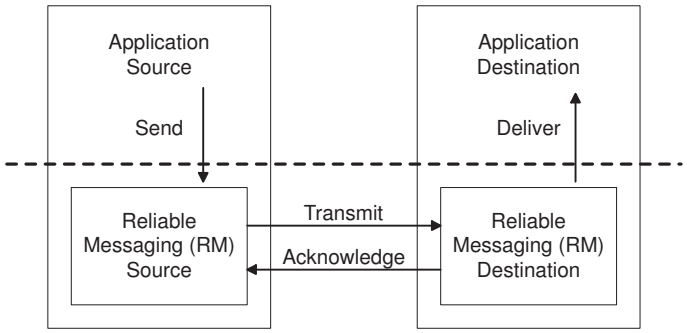

Abbildung 2.52: WS-ReliableMessaging Modell

- **WS-Reliability** Die *WS-Reliability*-Spezifikation liegt in der Version 1.1 [14] beim *WS-Reliable- Messaging (WSRM) Technical Committee* vor und beschreibt einen ähnlichen Ansatz zur Gewährleistung der zuverlässigen Nachrichtenübertragung wie die Spezifikation WS-ReliableMessaging.

- **WS-Inspection** Mit der *WS-Inspection*-Spezifikation [46] können WSDL-Beschreibungen und UDDI-Einträge für einen Web Service innerhalb einer einheitlichen Form zur Verfügung gestellt werden. Dazu werden Referenzen auf diese Dokumente innerhalb eines `<wsil:inspection>`-Dokuments aufgelistet. Diese *Inspection*-Dokumente können nun am Punkt des Angebotes, also des Web Service, veröffentlicht werden oder mittels verschiedener Medien, wie beispielsweise HTML, referenziert werden.

- **BPEL4WS** Die *Business Process Execution Language for Web Services (BPEL4WS)* [51] erlaubt mit Web Services *Workflows* aufzubauen und so komplexe Geschäftsprozessen abzubilden.

- **WS-Eventing** Die *WS-Eventing*-Spezifikation [52] definiert eine Reihe von Operationen für die asynchrone Nachrichtenübertragung. Ein Web Service-Nutzer kann mittels dieser Operationen die Informationen oder Funktionen, die ein Web Service bereitstellt, abonnieren und bekommt diese Angebote in festgelegten zeitlichen Abständen zugestellt. WS-Eventing kann somit beispielsweise zur Umsetzung eines Newsletter-Service eingesetzt werden.

2.9 Java ME

Das Referenzszenario des SicAri-Projekts sieht den Zugriff auf die Plattform durch ein mobiles Endgerät vor. Einem Flugzeugmechaniker soll es möglich sein, sich mit seinem PDA und eingesteckter Smart Card mit der SicAri-Plattform zu verbinden. Damit soll er in der Lage sein, für seine Arbeit benötigte Informationen direkt von entfernten Diensten abzurufen bzw. überprüfte Vorgänge direkt zu protokollieren und diese Protokolle signiert an die entsprechenden Dienste zu senden.

2.9.1 Übersicht über die Java ME

Die *Java Micro Edition (Java ME)* erlaubt es, die Vorteile von Java auch auf Geräten auszunutzen, die nur über geringe Hardwareressourcen verfügen. Diese Geräte unterscheiden sich im Wesentlichen in den Bereichen Displaygröße, Prozessorleistung, Speicherkapazität, Ein-/Ausgabe-Techniken sowie in der Bandbreite der Netzwerkverbindung. Um diesen verschiedenen Hardwareausstattungen zu begegnen, kann die Java ME individuell konfiguriert werden. Dazu verwendet die Java ME Konfigurationen und Profile, die in Kombination eine auf das jeweilige Gerät angepasste Java-Plattform bilden können. In Tabelle 2.10 & 2.11 sind diese Konfigurationen und Profile mit dem jeweiligen *Java Specification Request (JSR)*[19] aufgeführt. Zusätzlich kann die Java ME durch verschiedene optionale Pakete erweitert werden.
Konfigurationen bilden die Basiselemente des Java ME-Konzepts und beschreiben die verfügbaren Grundfunktionen einer virtuellen Maschine für eine Geräteklasse. Zur Zeit existieren die *Connected Limited Device Configuration (CLDC)* und die *Connected Device Configuration (CDC)*. Beide bestehen jeweils aus einer *Virtual Maschine (VM)* sowie einer geringen Anzahl von Java-Paketen. Die VM der CLDC wird auch als *Kilobyte Virtual Machine (KVM)* bezeichnet.
Profile setzen direkt auf den Konfigurationen auf und erweitern diese. Wie die Konfigurationen Geräte in unterschiedliche Klassen einteilen, kann durch den Einsatz der Profile auch auf die weiteren Unterschiede der Geräte einer Klasse eingegangen werden. Die Java-Plattform kann deshalb individuell auf die Hardware-Eigenschaften eines Gerätes zugeschnitten werden.

2.9.1.1 CLDC

Die *Connected Limited Device Configuration (CLDC)* ist für Endgeräte mit geringen Hardwareressourcen konzipiert. Dazu gehören beispielsweise Geräte wie Mobiltelefone, Pager und einfache *Personal Digital Assistants (PDA)*. Zur Ausführung verlangt die CLDC nur 160 KB Speicherkapazität. Die CLDC existiert in zwei Versionen. Die Version 1.1 [97] unterscheidet sich zur Version 1.0 [96], neben kleineren Verbesserungen in der Implementierung und der Beseitigung von Fehlern, hauptsächlich durch die zusätzliche Unterstützung der Datentypen Float und Double sowie die Möglichkeit des Arbeitens mit schwachen Referenzen (*weak references*). Durch diese Erweiterungen fordert die CLDC-Spezifikation eine Speicherkapazität von mindestens 192 KB.[98]
Innerhalb der CLDC ist derzeit nur das Profil *Mobile Information Device Profile (MIDP)* bekannt. Es existieren die Versionen 1.0 und 2.1. Außerdem ist eine Version 3.0 in Arbeit. Das MIDP ermöglicht die Ausführung so genannter MIDlets. MIDlets sind Anwendungen, die vergleichbar mit Applets sind. Sie werden im Gegensatz zu Applets nicht durch einen Webbrowser

[19] Ein JSR definiert neue Java-APIs bzw. spezifiziert Anforderungen für die Erweiterung oder Revision von bestehenden Java-APIs

gesteuert und kontrolliert, sondern durch eine spezielle Software. Diese wird als *Application Management Software (AMS)* bezeichnet und ist abhängig von dem Gerät, auf dem das MIDlet ausgeführt werden soll. Die AMS ist dafür zuständig, dass das MIDlet zu jedem Zeitpunkt seiner Ausführung unterbrochen werden kann. Dies ist notwendig damit das Gerät auf beispielsweise einen ankommenden Anruf reagieren kann.[37]
Der Paketumfang der CLDC sowie dessen Profile umfassen eine Teilmenge der Pakete der Java Standard Edition Version 1.3.1 (Java SE) (siehe Kapitel 2.9.3) mit jeweils spezifischen Erweiterungen, die im Namensraum `javax.microedition` zusammengefasst sind. Hier werden entsprechende APIs für die Erstellungen von Benutzeroberflächen, die Entwicklung von MIDlets, Funktionen für das Speichern und Laden von Daten, das Arbeiten mit Audio- und Videodaten sowie speziellen APIs für die Entwicklung von Spielen bereitgestellt. Außerdem definieren die Komponenten dieses Namensraums auch Ein- und Ausgabefunktionen sowie Netzwerkfunktionen und das sogenannte *Generic Connection Framework (GCF)*, das in Kapitel 2.9.2 vorgestellt wird.

CLDC	Version	JSR
Connected Limited Device Configuration	1.0a	JSR 30
Connected Limited Device Configuration	1.1	JSR 139
Mobile Information Device Profile	1.0	JSR 37
Mobile Information Device Profile	2.1	JSR 118

Tabelle 2.10: Java ME - Aufbau der Connected Limited Device Configuration

2.9.1.2 CDC

Die *Connected Device Configuration (CDC)* liegt derzeit in der Version 1.1.2 vor [101] und ist im Gegensatz zur CLDC für Geräte mit größerer Speicherkapazität, schnelleren Prozessoren und größerer Netzwerkbandbreite entwickelt worden. Hierzu zählen zum Beispiel Highend-PDAs, TV-Set-Top-Boxen, Smartphones oder auch Fahrzeugnavigationssysteme. Die VM der CDC unterstützt alle Eigenschaften der VM, wie sie unter der Java SE bekannt ist.
Im Gegensatz zu KVM der CLDC benötigt diese virtuelle Maschine allerdings eine Speicherkapazität von mindestens 2 MB Arbeitsspeicher sowie 2,5 MB permanenten Speicher. In Abhängigkeit der eingesetzten Profile kann dieser Wert noch ansteigen. Der Paketumfang der CDC ist im Vergleich zur CLDC erheblich erweitert worden. Es werden jetzt mehr Pakete aus der Java SE unterstützt. Die einzelnen Bestandteile der CDC sind in Tabelle 2.11 aufgeführt.[100]
Innerhalb der CDC existieren die Profile *Foundation Profile (FP)*, *Personal Basis Profile (PBP)* und *Personal Profile (PP)*. Alle drei Profile liegen derzeit in der Version 1.1.2 vor. Durch diese Profile kann der Funktionsumfang aufbauend auf der CDC stark erweitert werden.
Die CDC und ihre Profile sind aufbauend auf der Java SE 1.4.2 entwickelt worden und bilden eine Teilmenge dieser. Die Pakete der CDC umfassen verschiedene Pakete der Java SE jedoch meist mit geringerem Umfang. Erst durch das FP werden diese Pakete in ihrem aus der Java SE bekannten kompletten Funktionsumfang unterstützt. Zusätzlich erweitert das FP den Funktionsumfang um weitere Pakete. Mit dem FP sind jedoch noch keine grafischen Oberflächen realisierbar. Erst die auf dem FP aufsetzenden Profile PBP und PP ermöglichen die Entwicklung von einfachen und komplexen Oberflächen. Dies wird durch die fast komplette Unterstützung der `java.awt`-Pakete möglich. Zusätzlich können durch die Verfügbarkeit des `java.applet`-Pakets innerhalb des PP auch Applets entwickelt werden. Als weitere wichtige Erweiterung

bieten die Profile PBP und PP die Möglichkeit, so genannte *Xlets* zu entwickeln. Xlets sind vergleichbar mit MIDlets oder auch Applets und werden wie MIDlets durch eine AMS gesteuert. Sie werden daher auch als Managed Applications bezeichnet. [37]
Das Xlet Framework und alle anderen Erweiterungen der CDC sind, wie auch schon in der CLDC, im Namensraum `javax.microedition` zusammengefasst. Für eine genaue Übersicht der einzelnen Funktionsunterschiede zwischen der CDC und der Java SE sei auf das Kapitel 2.9.3 verwiesen.

CDC	Version	JSR
Connected Device Configuration	1.1.2	JSR 218
Foundation Profile	1.1.2	JSR 219
Personal Basis Profile	1.1.2	JSR 217
Personal Profile	1.1.2	JSR 216

Tabelle 2.11: Java ME - Aufbau der Connected Device Configuration

2.9.1.3 Optionale APIs

Die Java ME besteht aus weiteren optionalen Paketen, die in Tabelle 2.12 aufgeführt sind.

Optionale Pakete	Version	JSR
Java APIs for Bluetooth (Bluetooth, OBEX)	1.1	JSR 82
Content Handler API	1.0	JSR 211
Mobile Media API	1.2	JSR 135
Java Binding for the OpenGL ES	1.0	JSR 239
Java ME Web Services-Spezifikation (JAXP, JAX-RPC)	1.0	JSR 172
Security and Trust Services APIs (SATSA): ADPU, JCRMI, PKI, CRYPTO	1.0	JSR 177
Security (JSSE, JCE, JAAS)	1.0	JSR 219
Advanced Graphics and User Interface (AGUI)	1.0	JSR 209
RMI	1.0	JSR 66
JDBC	1.0	JSR 169
Java TV API	1.1	JSR 927
PDA Optional Packages	1.0	JSR 75
Mobile 3D Graphics API	1.0	JSR 184
Location API	1.0	JSR 179
Mobile Internationalization API	1.0	JSR 238
Payment API	1.0	JSR 229
SIP API	1.0	JSR 180
Advanced Multimedia Supplements	1.0	JSR 234
Scalable 2D Vector Graphics API	1.0	JSR 226

Tabelle 2.12: Java ME - Optionale Pakete

2.9.2 Generic Connection Framework

Bei diesem Framework handelt es sich um eine Vereinheitlichung der Zugriffe auf unterschiedliche Verbindungstypen. Es besteht nur aus einer einzigen Klasse und besitzt eine Vielzahl von Schnittstellen für die verschiedenen Verbindungstypen. Das Framework wurde für die Java ME entwickelt, um die verschiedenen Verbindungen zu vereinheitlichen, die unter der Java SE mit den Paketen `java.io` und `java.net` realisiert werden. Dies ist notwendig, da Java ME-Geräte nur über begrenzte Hardware-Ressourcen verfügen. Die Portierung der genannten Pakete auf die Java ME-Plattform ist zwar prinzipiell möglich, die Nutzung würde jedoch zu viele Ressourcen auf dem Gerät verbrauchen. Unter der Java SE werden beispielsweise die Klassen `Socket` und `Datagram` des Pakets `java.net` genutzt, um Netzwerkverbindungen aufzubauen. Innerhalb der Java ME wird dies generell über die Klasse `Connector` abgewickelt. Durch den Aufruf der statischen Methode `open(String name)` wird bei erfolgreichem Verbindungsaufbau eine Instanz des Typs `Connection` zurückgeliefert. Nach einem *Casting* in beispielsweise eine Instanz des Typs `DatagramConnection`, `StreamConnection` oder `HttpsConnection` kann eine entsprechende Verbindung genutzt werden. Welcher Verbindungstyp geöffnet werden soll, wird durch den Parameter der `open` Methode definiert.[79]

2.9.3 Vergleich der Java ME mit der Java SE

Zum Verständnis der Java ME ist es sehr hilfreich, wenn diese mit der Java SE verglichen wird. Dadurch wird leicht und schnell ersichtlich, aus welchen Paketen die Java ME besteht. Außerdem wird deutlich, welche Pakete nicht aus der Java SE übernommen wurden. Damit kann der Funktionsumfang der Java ME schnell überschaut werden und entschieden werden, welche Konfiguration in Verbindung mit welchen Profilen für den jeweiligen Anwendungsfalls benötigt wird.
Die CLDC basiert auf der Java SE 1.3.1. Tabelle 2.13 zeigt die Unterschiede und Gemeinsamkeiten der CLDC und seiner Profile zu der Java SE. Für eine noch detailliertere Übersicht mit Angabe der einzelnen Paket-Bestandteile, sei auf das Kapitel 6.2 der jeweiligen Spezifikation [96, 97] verwiesen.
Leere Felder verdeutlichen, dass das angegebene Paket innerhalb der Konfiguration bzw. des Profils nicht unterstützt wird. Grau hinterlegte Felder verdeutlichen, dass nur ein Teil der Funktionen des jeweiligen Pakets der Java SE unterstützt wird. Ein schwarzes Feld bedeutet, dass der komplette Umfang des Pakets in der Java ME unterstützt wird. Teilweise enthalten einige Felder Informationen zu Paketen, die optional installiert werden können, um die jeweilige Funktionalität zu erhalten.
Aus der Tabelle wird ersichtlich, dass der Funktionsumfang der CLDC im Vergleich zur Java SE stark eingeschränkt ist. Weite Teile der Java SE sind in der CLDC nicht vorhanden. Auch durch die Einbindung der MIDP-Profile lässt sich der Funktionsumfang nur geringfügig erweitern.

Package	CLDC 1.0a	MIDP 1.0	CLDC 1.1	MIDP 2.1
java.applet				
java.awt				
java.beans				
java.io				
java.lang				
java.lang.ref				
java.lang.reflect				

java.math				
java.net				
java.rmi				
java.security	Durch optionales Paket SATSA			
java.sql				
java.text				
java.util				
java.util.jar				
java.util.zip				
javax.accessibility				
javax.naming				
javax.rmi				
javax.sound				
javax.swing				
javax.transaction				
org.omg				

Tabelle 2.13: Java ME - Vergleich CLDC und Java SE 1.3.1

Die CDC basiert auf der Java SE 1.4.2. In Tabelle 2.14 werden die einzelnen Unterschiede zur CDC und seiner Profile dargestellt. Die Übersicht zeigt, dass durch Kombination der CDC mit den Profilen FP, PBP und PP und verschiedener optionaler Pakete fast der komplette Funktionsumfang der Java SE abgebildet werden kann.

Package	CDC 1.1.2	FP 1.1.2	PBP 1.1.2	PP 1.1.2
java.applet				
java.awt				
java.awt.color				
java.awt.datatransfer				
java.awt.dnd, java.awt.-geom,java.awt.im.spi, java.awt.image.-renderable,java.awt.print				
java.awt.event				
java.awt.font				
java.awt.im				
java.awt.image				
java.beans				
java.beans.beancontext				
java.io				
java.lang				
java.lang.ref				
java.lang.reflect				
java.math				
java.net				
java.nio				

java.rmi				
java.rmi.activation,java.-rmi.dgc, java.rmi.server				
java.rmi.registry				
java.security				
java.security.acl				
java.security.cert				
java.security.interfaces				
java.security.spec				
java.sql	Durch optionales Paket SQL			
java.text				
java.util				
java.util.jar				
java.util.logging,java.-util.prefs,java.util.regex				
java.util.zip				
javax.accessibility				
javax.crypto	Durch optionales Paket JCE			
javax.net, javax.net.ssl, javax.security.cert	Durch optionales Paket JSSE			
javax.security.auth	Durch optionales Paket JAAS[20]			
javax.security.auth.x500				
javax.imageio	Durch optionales Paket AGUI			
javax.naming				
javax.print				
javax.rmi				
javax.sound				
javax.sql	Durch optionales Paket SQL			
javax.swing	Durch optionales Paket AGUI			
javax.transaction				
javax.xml				
org.ietf				
org.omg				
org.w3c.dom				
org.xml				

Tabelle 2.14: Java ME - Vergleich CDC und Java SE 1.4.2

[20] ohne Package javax.security.auth.kerberos

Kapitel 3

SicAri

In diesem Kapitel wird die SicAri-Plattform vorgestellt. Zunächst wird ein Überblick über die Plattform gegeben und deren Anwendungsbereiche skizziert. Danach werden die Sicherheitsanforderungen an SicAri detailliert und schließlich die Architektur und Dienste der Plattform vorgestellt.

3.1 Überblick zur SicAri-Plattform

Für eine professionelle Nutzung heutiger Kommunikations- und Kollaborationsinfrastrukturen ist es notwendig, dass geeignete Sicherheitsmaßnahmen berücksichtigt werden. Die Sicherheitsmechanismen müssen dabei für den Benutzer einfach und ohne großen Aufwand zu benutzen sein. Das Hauptziel des SicAri-Projekts ist daher die Konzeption und Realisierung einer Sicherheitsplattform, die die ubiquitäre Nutzung des Internets ermöglicht und dabei die Sicherheitsfunktionen für den Benutzer in einer transparenten Weise integriert.
Die SicAri-Plattform stellt eine offene Middleware dar, deren Architektur von den damit zu realisierenden Anwendungsszenarien bestimmt ist. Bei der Konzeption der Architektur wurden dabei verschiedene Gesichtspunkte beachtet. Die sich daraus ergebenden Anforderungen an die Plattform sind in der Anforderungsspezifikation [75] festgelegt. Hierbei wurden zunächst besondere Anforderungen an den Entwurf gestellt. Neben der Verwendung von anerkannten Entwurfsmustern sowie der Nutzung von Standards und Portabilitätsanforderungen wurden hier Anforderungen an die Plattform-Infrastruktur (Dienstlokalisierung, Kommunikation, Kommunikationsprotokolle) und die lokale Architektur (Programmiersprache, Schnittstellen, Wiederverwendung, ...) gestellt.
Die Sicherheitsanforderungen und funktionalen Anforderungen stellen einen weiteren Aspekt bei der Konzeption der Plattform dar. Die Sicherheitsanforderungen werden gesondert in Kapitel 3.3 vorgestellt. Zu den funktionalen Anforderungen gehören spezielle Anforderungen an die Konfigurationsmöglichkeiten, die Nebenläufigkeit und die Ereignisbehandlung.
Die Anforderungsspezifikation [75] von SicAri enthält außerdem noch Anforderungen an die externen Schnittstellen und die Festlegung von verschiedenen Qualitätsaspekten, die beispielsweise die Fehlerbehandlung, Wartbarkeit und Protokollierung betreffen.
Abbildung 3.1 gibt einen einfachen Überblick über die Architektur der SicAri-Plattform. Die oberste Ebene stellt die Anwendungsebene bereit. Benutzer können direkt auf die in dieser Ebene vorhandenen Anwendungen zugreifen. Die Anwendungen ihrerseits nutzen die Services, die in der Dienstebene der Plattform bereitgestellt werden. Hier werden Services zur Verfügung gestellt, die es beispielsweise ermöglichen, auf Datenbanken, externe Applikationen und Legacy-

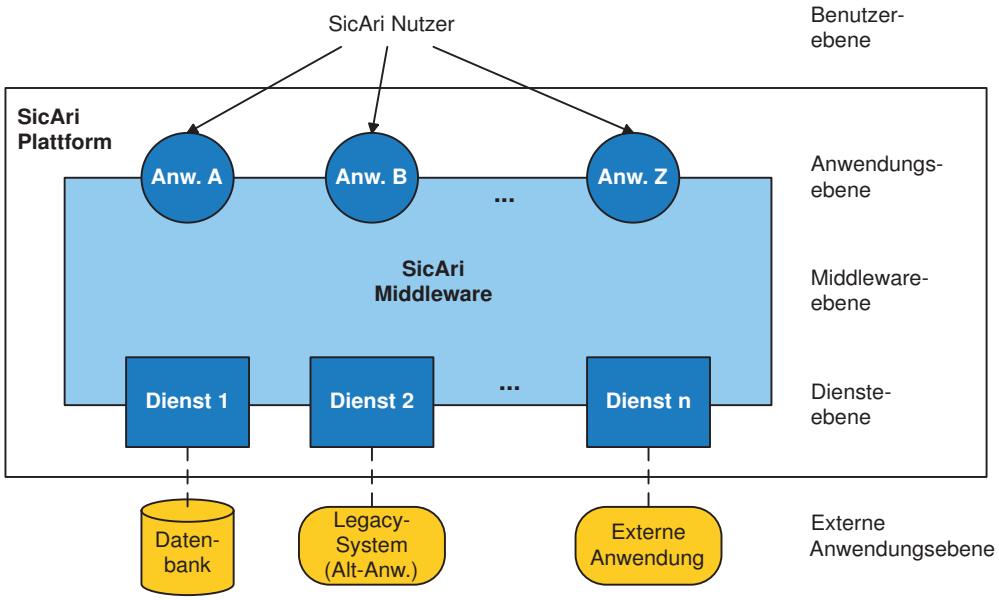

Abbildung 3.1: High-Level Architektur der SicAri-Plattform

Anwendungen zuzugreifen. Die Service-Ebene stellt zudem wichtige Sicherheitsdienste bereit. Darunter fallen Dienste für die Authentifikation und Zugriffskontrolle. Die Middleware-Ebene ist für eine transparente Integration der Dienste und Anwendungen verantwortlich und ermöglicht deren sichere Kommunikation untereinander.

3.2 Anwendungsbereiche der Plattform

Die Hauptfunktion der SicAri-Plattform ist die Bereitstellung von Schnittstellen für die Applikationen der Anwendungsebene. Diese Schnittstellen erlauben es den Anwendungen auf die Dienste der Service-Ebene zuzugreifen. Zusätzlich kann mit der Plattform eine Infrastruktur aufgebaut werden, die die Kommunikation zwischen verteilten SicAri-Komponenten ermöglicht.
Die Plattform ist nahtlos in die bestehende IT-Infrastruktur integrierbar. Anpassungen an bestehenden Systemen müssen nicht vorgenommen werden. Vielmehr bietet sie die Möglichkeit, bestehende Systeme in die Plattform einzubinden und so neue Anwendungsbereiche zu realisieren. Durch den modularen Aufbau, die leichte Erweiterbarkeit und die Flexibilität der Plattform, ist es zudem schnell und einfach möglich, neue Dienste zu integrieren, bestehende SicAri-Dienste anzupassen und die Plattform zu erweitern. Die Plattform bietet ein hohes Maß an Skalierbarkeit, da eine Lastverteilung durch die Installation von mehreren SicAri-Instanzen möglich ist. Die verschiedenen Instanzen kommunizieren dabei über die SicAri-Middleware und ermöglichen so den Aufbau einer Infrastruktur bestehend aus mehreren SicAri-Plattformen.
Alle Sicherheitsaspekte einer SicAri-Plattform werden durch eine Sicherheitsrichtlinie (*Security Policy*) bestimmt. Interagieren mehrere Plattformen in einer SicAri-Infrastruktur, so teilen diese Plattformen die gleiche Sicherheitsrichtlinie. Jede Plattform beinhaltet hierfür eine so genannte *Policy Enforcement Point (PEP)*-Komponente, die für die Durchsetzung einer festgelegten

Richtlinie zuständig ist. Zugriffe auf lokale oder entfernte Dienste der Plattform werden bezüglich der Sicherheitsrichtlinie überprüft. Hierbei werden verschiedene Aspekte, wie zum Beispiel die aktuelle Benutzersitzung, die Rolle des Benutzers und dessen Zugriffsberechtigungen, berücksichtigt.
Die SicAri-Plattform und ihre Komponenten basieren auf der Softwaretechnologie und Programmiersprache Java. *SUN Microsystems*[1] stellt für verschiedene Einsatzbereiche unterschiedliche Java-Editionen bzw. Plattformen zur Verfügung. Die *Java Enterprise Edition (Java EE)* ist für den Einsatz in skalierbaren und verteilten Umgebungen konzipiert und wird meist auf Großrechnern und Servern eingesetzt. Für den Einsatz auf üblichen PCs existiert die *Java Standard Edition (Java SE)*. Die *Java Micro Edition (Java ME)* ist für den mobilen Einsatz konzipiert und erlaubt es Java auch auf mobilen Geräten mit nur geringen Hardwareressourcen einzusetzen. Durch das breite Anwendungsspektrum und die weite Verbreitung der Java-Technologie auf allen gängigen Betriebssystemen, kann auch die SicAri-Plattform von diesen Vorteilen profitieren. Derzeit setzt SicAri mindestens die Java SE voraus, da die Java ME noch keine Unterstützung für viele kryptografische Funktionen zur Umsetzung der spezifizierten Sicherheitsmechanismen bietet.

3.3 Sicherheitsanforderungen

Um die Sicherheitsziele der SicAri-Plattform umsetzen zu können, müssen eine Reihe von Sicherheitsfunktionen implementiert werden. Im Folgenden werden diese Funktionalitäten erläutert.

- Erstellung von Überwachungsprotokollen
  Alle relevanten Sicherheitsereignisse müssen von der Plattform in einer Überwachungsdatenbank protokolliert werden. Hierbei müssen auch alle notwendigen Informationen, wie beispielsweise der Zeitpunkt des Ereignisses und der betroffene Benutzer, in der Datenbank gesichert werden.

- Zugriffsbeschränkungen auf die Überwachungsprotokolle
  Die Überwachungsprotokolle beinhalten sensible Daten, die nur von autorisierten Benutzern, wie beispielsweise dem Plattform-Administrator, eingesehen werden dürfen. Der Zugriff auf diese Daten muss daher entsprechend geschützt werden.

- Selektierbarkeit der Überwachungsprotokolle
  Der Person, die für die Überwachung der Sicherheit zuständig ist, müssen eine Menge von geeigneten Such- und Auswahlmechanismen zur Verfügung gestellt werden, die die Analyse der Überwachungsprotokolle vereinfachen.

- Bereitstellung kryptografischer Grundfunktionen
  Für die Ver- und Entschlüsselung von Daten sowie die Erstellung und Verifikation von digitalen Signaturen müssen in der Service-Ebene entsprechende Dienste bereitgestellt werden. Die Applikationen der Anwendungsebene können diese Funktionen nutzen, um verschiedene Schutzziele zu gewährleisten.

- Rollenbasierende Zugriffskontrolle
  Das *Role-based access control (RBAC)*-Konzept erlaubt es auf einfache und effiziente

[1] http://www.sun.com

Weise, basierend auf der festgelegten Rolle eines Benutzers, Zugriffskontrollen durchzuführen.

- Informationsflusskontrolle und Filterung
  Benutzerbestimmte Zugriffskontrollmechanismen (vgl. Kapitel 2.2.1) sind für die SicAri-Plattform nicht ausreichend. Deshalb müssen auch systembestimmte Zugriffskontrollen zur Verfügung gestellt werden, die die Kontrolle und Filterung von Informationen ermöglichen.

- Erkennung von fehlerhaften Anmeldeversuchen
  Versuche den Authentifikationsmechanismus zu brechen müssen durch geeignete Maßnahmen verhindert werden. Der Mechanismus muss deshalb eine Benutzerkennung ggf. sperren, wenn zu viele fehlgeschlagene Authentifikationsversuche durchgeführt wurden. Der Mechanismus minimiert dadurch die Wahrscheinlichkeit für das Auftreten von unautorisierten Zugriffen durch mögliche Angreifer.

- Unterstützung von Passwort- und Public Key basierter Authentifikation
  Die Plattform muss die Möglichkeit bieten, sich mit Passwort- und Public Key-basierten Authentifikationsmechanismen zu authentifizieren.

- Management von Benutzer- und Administratorrollen
  Die Administration von Rollen ist in verteilten Systemen eine nicht zu unterschätzende Aufgabe. Die Plattform muss deshalb entsprechende Dienste bereitstellen, die das Rollenmanagement ermöglichen.

- Selbsttest von Services
  Um die Funktionalität eines Services gewährleisten zu können, darf ein Service eine zusätzliche Selbsttest Funktionalität bereitstellen. Diese kann regelmäßig oder auf Anfrage ausgeführt werden und ermöglicht die Überprüfung der korrekten Funktionsweise des Service.

- Management von Sicherheitsfunktionen und deren Parameter
  Die SicAri-Plattform kann in unterschiedlichen Anwendungsszenarien eingesetzt werden, die jeweils unterschiedliche Sicherheitsanforderungen besitzen. Daher muss die Plattform entsprechend konfigurierbar sein. Durch den Einsatz von Sicherheitsrichtlinien muss das Verhalten der Plattform und deren Dienste entsprechend den Sicherheitsanforderungen bestimmt werden können.

- Entwerten von Sicherheitsmerkmalen
  Sicherheitsmerkmale (repräsentiert durch ein Security Token), wie beispielsweise Zertifikate, sind meist nur einen bestimmten Zeitraum gültig oder können ihre Gültigkeit komplett verlieren, wenn sie unberechtigt eingesetzt werden. Ein autorisierter Administrator muss in der Lage sein, solche Sicherheitsmerkmale eines Benutzers zu entwerten und damit ungültig zu machen.

- Vertrauenswürdiger Plattform-Administrator
  Der Plattform-Administrator muss vertrauenswürdig sein. Er muss eine eindeutige Benutzerkennung und ein gültiges Zertifikat besitzen. Außerdem muss die Plattform von einem vertrauenswürdigen Administrator gestartet und administriert werden. Der Administrator ist damit Dreh- und Angelpunkt für die erfolgreiche Umsetzung der Sicherheitsziele.

- Abwehr von böswilligen Diensten
  Die Plattform muss Mechanismen bereitstellen, um die Ausführung von schadhaften oder böswilligen Diensten zu verhindern. Dazu wird die Benutzerkennung des Plattform-Administrators mit der Plattform-ID zu einer eindeutigen Kennung kombiniert. Von der Plattform darf nur signierter Code ausgeführt werden. Diese Vorgehensweise erlaubt es, die Ausführung von böswilligen Diensten zu verhindern, da diese normalerweise keine gültige Signatur besitzen.

Diese funktionalen Sicherheitsanforderungen sind in der SicAri-Anforderungsspezifikation PF2 [75] detailliert spezifiziert. Sie wurden unter zu Hilfenahme der im zweiten Teil der Common Criteria Version v2.3 (vgl. Kapitel 2.2.4) definierten Kriterien erstellt. Tabelle 3.1 listet die gestellten Sicherheitsanforderungen auf. Eine genaue Beschreibung der einzelnen Anforderungen kann der Spezifikation ([75], Kapitel 3) entnommen werden.

Komponente	**Kurzbezeichnung**	**Funktionale Elemente**
Audit Data Generation	FAU_GEN.1	FAU_GEN.1.1
		FAU_GEN.1.2
User Identity Generation	FAU_GEN.2	FAU_GEN.2.1
Security Audit Review	FAU_SAR.1	FAU_SAR.1.1
		FAU_SAR.1.2
Restricted Audit Review	FAU_SAR.2	FAU_SAR.2.1
Selectable Audit Review	FAU_SAR.3	FAU_SAR.3.1
Selective Audit	FAU_SEL.1	FAU_SEL.1.1
Permanent Audit Trail Storage	FAU_STG.1	FAU_STG.1.1
		FAU_STG.1.2
Prevention of audit data loss	FAU_STG.4	FAU_STG.4.1
Cryptographic operation	FCS_COP.1	FCS_COP.1.1
Subset Access Control	FDP_ACC.1	FDP_ACC.1.1
Security Attribute Based Access Control	FDP_ACF.1	FDP_ACF.1.1
		FDP_ACF.1.2
		FDP_ACF.1.3
		FDP_ACF.1.4
Subset information flow control	FDP_IFC.1	FDP_IFC.1.1
Simple security attributes	FDP_IFF.1	FDP_IFF.1.1
		FDP_IFF.1.2
		FDP_IFF.1.6
Subset residual information protection	FDP_RIP.1	FDP_RIP.1.1
Authentication failure handling	FIA_AFL.1	FIA_AFL.1.1
		FIA_AFL.1.2
User Attribute Definition	FIA_ATD.1	FIA_ATD.1.1
Timing of authentication	FIA_UAU.1	FIA_UAU.1.2
User Authentication before any action	FIA_UAU.2	FIA_UAU.2.1
Multiple authentication mechanisms	FIA_UAU.5	FIA_UAU.5.1
		FIA_UAU.5.2
Protected authentication feedback	FIA_UAU.7	FIA_UAU.7.1
Timing of identification	FIA_UID.1	FIA_UID.1.2
User Identification before any action	FIA_UID.2	FIA_UID.2.1
User-Subject Binding	FIA_USB.1	FIA_USB.1.1

Komponente	Kurzbezeichnung	Funktionale Elemente
Management of security functions behaviour	FMT_MOF.1	FMT_MOF.1.1
Management of Security Attributes	FMT_MSA.1	FMT_MSA.1.1
Secure Security Attributes	FMT_MSA.2	FMT_MSA.2.1
Static attribute initialisation	FMT_MSA.3	FMT_MSA.3.1
		FMT_MSA.3.2
Management of TSF Data	FMT_MTD.1	FMT_MTD.1.1
Management of limits on TSF data	FMT_MTD.2	FMT_MTD.2.1
		FMT_MTD.2.2
Secure TSF Data	FMT_MTD.3	FMT_MTD.3.1
Revocation	FMT_REV.1	FMT_REV.1.1
		FMT_REV.1.2
Specification of Management Functions	FMT_SMF.1	FMT_SMF.1.1
Security roles	FMT_SMR.1	FMT_SMR.1.1
		FMT_SMR.1.2
Restrictions on Security Roles	FMT_SMR.2	FMT_SMR.2.1
		FMT_SMR.2.2
		FMT_SMR.2.3
Abstract Machine Testing	FPT_AMT.1	FPT_AMT.1.1
Failure with preservation of Secure State	FPT_FLS.1	FPT_FLS.1.1
Manual Recovery	FPT_RCV.1	FPT_RCV.1.1
Function Recovery	FPT_RCV.4	FPT_RCV.4.1
Non-bypassability of the TSP	FPT_RVM.1	FPT_RVM.1.1
TSF Domain Separation	FPT_SEP.1	FPT_SEP.1.1
		FPT_SEP.1.2
Reliable Time Stamps	FPT_STM.1	FPT_STM.1.1
TSF Self Test	FPT_TST.1	FPT_TST.1.1
		FPT_TST.1.2
		FPT_TST.1.3
Limitation on Scope of Selectable Attributes	FTA_LSA.1	FTA_LSA.1.1
TOE Session Establishment	FTA_TSE.1	FTA_TSE.1.1

Tabelle 3.1: Funktionale Sicherheitsanforderungen der SicAri-Plattform

3.4 Architektur der Plattform

Die Architektur der Plattform wird einerseits durch die Anforderungsspezifikation der Plattform [75] und andererseits durch die Spezifikation der Anforderungen an die Sicherheitsrichtlinie [92] bestimmt. In den Anforderungen an die Sicherheitsrichtlinie ist festgelegt, wie Sicherheitsrichtlinien integriert, administriert, verifiziert, spezifiziert und durchgesetzt werden.
Ein Überblick über die sich daraus ergebende Architektur der Plattform (vgl. Abbildung 3.2) soll nun anhand eines konkreten Anwendungsfalls gegeben werden. In diesem Beispielfall meldet sich ein Benutzer an einer Plattform an und nutzt verschiedene Anwendungen, die in der Plattform installiert sind:

1. Zunächst muss sich der Benutzer über die SicAri-Shell (siehe Kapitel 3.4.1) an der Platt-

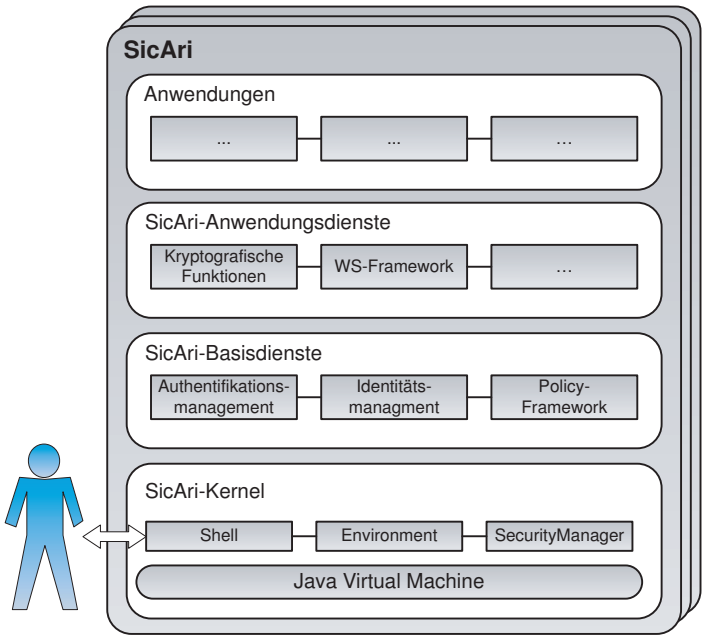

Abbildung 3.2: Architektur der SicAri-Plattform

form anmelden. Dazu wird im Rahmen des Authentifikationsmanagements (siehe Kapitel 3.5.3) die Identität des Benutzers mit Hilfe eines Passworts oder einer Smart Card überprüft und über das Identitätsmanagement (siehe Kapitel 3.5.4) verifiziert. Die Authentifikation wird entweder durch den lokalen Authentifikations-Service der Plattform oder über den globalen Authentifikationsmanager der SicAri-Infrastruktur abgewickelt (siehe Kapitel 3.5.3).

2. Nach dem der Benutzer erfolgreich authentifiziert wurde, wird für den Benutzer ein Sicherheitskontext (siehe Kapitel 3.4.1.3) erstellt. Der Benutzer agiert nun im Rahmen seines Sicherheitskontextes mit der Plattform.

3. Bei jeder Aktion des Benutzers wird durch den `SecurityManager` des SicAri-Kernel überprüft, ob die gewünschte Aktion ausgeführt werden darf. Dies wird anhand des Sicherheitskontextes des Benutzers festgestellt. Der `SecurityManager` konsultiert dazu das Policy-Framework (siehe Kapitel 3.5.1). Mit Hilfe des Policy-Framework werden innerhalb von SicAri Sicherheitsrichtlinien definiert. Anhand der spezifizierten Sicherheitsrichtlinien, der Rolle des Benutzers und des Sicherheitskontext entscheidet eine spezielle Komponente des Policy-Framework, ob eine Aktion durchgeführt werden darf oder nicht.

4. Möchte ein Benutzer eine bestimmte Anwendung aufrufen, stellt das Environment die Schnittstelle für den Zugriff bereit. Hierfür überprüft das Environment, ob die aufgerufene Anwendung auf der lokalen Plattform existiert. Konnte die Anwendung gefunden werden, wird der Aufruf der Anwendung ggf. über einen Proxy (siehe Kapitel 3.4.1.2) durchgeführt.

   Ist dies nicht der Fall, überprüft das Environment automatisch, ob die Anwendung ggf. auf einer entfernten Plattform zur Verfügung steht. Hierfür konsultiert das Environment das SicAri-WS-Framework (siehe Kapitel 3.4.3.1). Dabei wird die SicAri-UDDI-Registry

anhand des definierten Anwendungspfads (siehe Kapitel 3.4.1.2) durchsucht. Konnte die Anwendung gefunden werden, wird automatisch ein entsprechender WS-Client generiert und im lokalen Environment publiziert. Der Aufruf des Benutzers wird nun automatisch über den WS-Client an die entfernte Anwendung weitergeleitet.

5. Die Kommunikation mit einer entfernten Anwendung wird durch die kryptografischen Funktionen (siehe Kapitel 3.6.2) der Plattform ggf. vor Verletzungen der Vertraulichkeit und Integrität geschützt.

Die in diesem Beispiel dargestellten Komponenten sind, wie in Abbildung 3.2 zu erkennen, in so genannte Basis- und Anwendungsdienste gruppiert. Diese Ebenen stellen für die Anwendungen der Benutzerebene verschiedene Dienste zur Verfügung.

3.4.1 SicAri-Kernel

Die Basis der SicAri-Plattform bildet der so genannte SicAri-Kernel, der auf der *Java Virtual Machine (JVM)* aufsetzt. Der Begriff *Kernel* wird innerhalb von SicAri als Synonym für den Begriff *Middleware* verwendet und besteht aus den drei Komponenten *Shell*, *Environment* und *Security Manager*. Zusammen ermöglichen diese Komponenten das Starten und Aktivieren von Diensten (*Bootstrapping*) sowie die Konfiguration und Verwaltung lokaler Dienste. Des Weiteren werden über den Kernel das Identitätsmanagement abgewickelt und die Sicherheitsrichtlinien durchgesetzt.

3.4.1.1 Shell

Die *Shell* stellt eine Interaktionsschicht mit der Plattform dar. Sie besitzt vergleichbar mit einer *UNIX-Shell* vordefinierte Kommandos, Variablen und eine festgelegte Syntax, die es erlaubt, Kontrollsequenzen, wie beispielsweise Schleifen und Bedingungen, zu nutzen sowie eigene Kommandos zu definieren.
Die Shell erlaubt es, Dienste zu starten, zu konfigurieren und diese zur Laufzeit zu überwachen. Die unterschiedlichen Kommandos der Shell können auch in einem Shell-Skript verwendet werden. So wird beispielsweise das Bootstrapping über ein Shell-Skript gesteuert.
Ein spezieller Service erlaubt es zudem, sich mit einem entsprechendem Client auf eine Shell einer entfernten Plattform anzumelden. Dazu stellt dieser Service einen auf Telnet basierenden Server bereit. Die Kommunikation wird dabei durch das SSL/TLS-Protokoll abgesichert und der Client durch die Option der Client-Authentifikation identifiziert und authentifiziert. Dadurch wird die entfernte Administration einer Plattform ermöglicht.

3.4.1.2 Environment

Das *Environment* ist für die Verwaltung der Dienste der Plattform zuständig. Das Environment stellt hierbei einen hierarchisch strukturierten Namensraum dar, in dem Dienste unter einem bestimmten Pfad publiziert werden.
Analog zu einem Dateisystem kann auch hier der Zugriff auf Dienste anhand von Zugriffsrechten beschränkt werden. Diese Rechte werden anhand der Pfade vergeben. SicAri unterscheidet hierbei Lesezugriffe (*lookup*) und zwei Arten von Schreibzugriffen (*publish* und *retract*).
Die Kommunikation zwischen Diensten untereinander, zwischen einem Dienst und dem Kernel sowie Anwendungen und Diensten wird über das Environment ermöglicht. Daher stellt das

Environment auch den geeignetsten Ort für die Überprüfung und Durchsetzung der Sicherheitsrichtlinien dar.
Das Environment ermöglicht außerdem Services mit Hilfe eines Stellvertreterobjekts vom Aufrufer des Services zu separieren. Beim Publizieren eines Dienstes kann ein Administrator einen so genannten *Dynamic Proxy* festlegen, der bei der Interaktion mit einem Service wichtige Aufgaben übernimmt und als Stellvertreter für den eigentlichen Dienst dient. SicAri stellt drei Proxy-Typen bereit, die alle die Schnittstelle `de.sicari.kernel.ProxyHandler` implementieren und jeweils den eigentlichen Dienst hinter dem Proxy verhüllen (*wrap*).

- Plain Proxy
  Ein so genannter *Plain Proxy* leitet Methodenaufrufe zu dem zugehörigen Service bzw. Service-Objekt innerhalb des gleichen *Threads*[2] weiter. Durch eine privilegiertes Signal kann der Proxy seine Referenz auf das zugehörige Service-Objekt entfernen und den Service für den *Garbage Collector* freigeben, falls keine weiteren Referenzen auf das Service-Objekt bestehen. Sollten weitere Zugriffe auf den Service erfolgen, so erzeugt der zugehörige Proxy eine entsprechende Fehlermeldung. Auf diese Weise kann ein Service sauber entfernt werden und ein inkonsistenter Zustand während eines Zugriffs verhindert werden.

- Asynchronous Proxy
  Der so genannte *Asynchronous Proxy* erzeugt für die Bearbeitung eines Methodenaufrufs jeweils einen eigenen Thread. Der Thread des Aufrufers wird dann solange blockiert, bis das Resultat der aufgerufenen Methode vorhanden ist bzw. ein festgelegter *Timeout* angelaufen ist.

- Security Proxy
  Der *Security Proxy* (Sicherheitsproxy) ist ein besonderer Proxy, der die automatische und transparente Durchsetzung der Sicherheitsrichtlinie ermöglicht. Beim Publizieren eines Dienstes wird zusätzlich immer automatisch ein Sicherheitsproxy im Environment installiert. Wird dies nicht gewünscht, so muss dies explizit angeben werden. Der Sicherheitsproxy dient einerseits dazu, den Aufruf von Methoden eines Dienstes zu autorisieren und andererseits die Methoden eines Dienstes unter der Kennung des Benutzers durchzuführen, der den Dienst ursprünglich publiziert hat. Der Sicherheitsproxy ist selbst ein so genannter *Wrapper* für einen der zuvor vorgestellten Proxies. Er stellt damit eine Art Hülle für einen *Plain Proxy* oder einen *Asynchronous Proxy* dar.

Der *Plain Proxy* und der *Asynchronous Proxy* stellen jeweils nur die öffentlichen Methoden eines Dienstes für den Zugriff zur Verfügung. Die Proxies reduzieren maßgeblich die Anfälligkeit der Dienste gegenüber *Denial of Service (DoS)*-Attacken und ermöglichen eine klare Trennung der Sicherheitskontexte zwischen Aufrufer und Aufgerufenem. Der *Security Proxy* realisiert die Zugriffskontrolle.

3.4.1.3 Sicherheitskontext

Neben den Sicherheitsmechanismen, die durch die *Shell* und das *Environment* realisiert werden, bietet der SicAri-Kernel einen impliziten und expliziten Sicherheitskontext (*Security Context*)

[2] Als Thread wird ein Teil eines Prozesses bezeichnet, der eine bestimmte Folge von Programmieranweisungen ausführt. Ein Thread teilt sich mit ggf. weiteren Threads eines Prozesses eine Reihe von Betriebsmitteln. Threads können daher direkt untereinander kommunizieren.

für die Dienste, die von der Shell gestartet wurden bzw. für die Benutzer, die sich an der Plattform angemeldet haben. Der Sicherheitskontext ist die Voraussetzung für die implizite Durchsetzung der Sicherheitsrichtlinie durch den SicAri-Sicherheitsmanager.

Einerseits können Dienste automatisch mit ihrem eigenen Sicherheitskontext gestartet werden. Dies geschieht durch einen eigenen `ClassLoader`[3] und ggf. in einer eigenen neuen `ThreadGroup`[4]. Generell werden Dienste von oder im Namen des Plattform-Administrators gestartet. Die Dienste erben damit automatisch den Sicherheitskontext des Administrators in Form von gewährten Genehmigungen. Mit diesem so genannten *Service Security Context* ist es möglich, die Rechte die ein Service besitzt dynamisch zu reglementieren.

Andererseits setzt der *Service Security Context* die Sicherheitsrichtlinie in Abhängigkeit des Benutzers der den Dienst aufruft durch (*Policy Enforcement*). Dazu delegiert er die dafür notwendige Entscheidung (*Policy Decision*) an ein spezielles Modul (vgl. Abbildung 3.3). Dieses Modul benötigt für die Entscheidung die ID des Benutzers, der diesen Service aufruft. Das Modul erhält dadurch Zugriff auf den zugehörigen Sicherheitskontext und kann daraufhin den Zugriff erlauben oder verweigern.

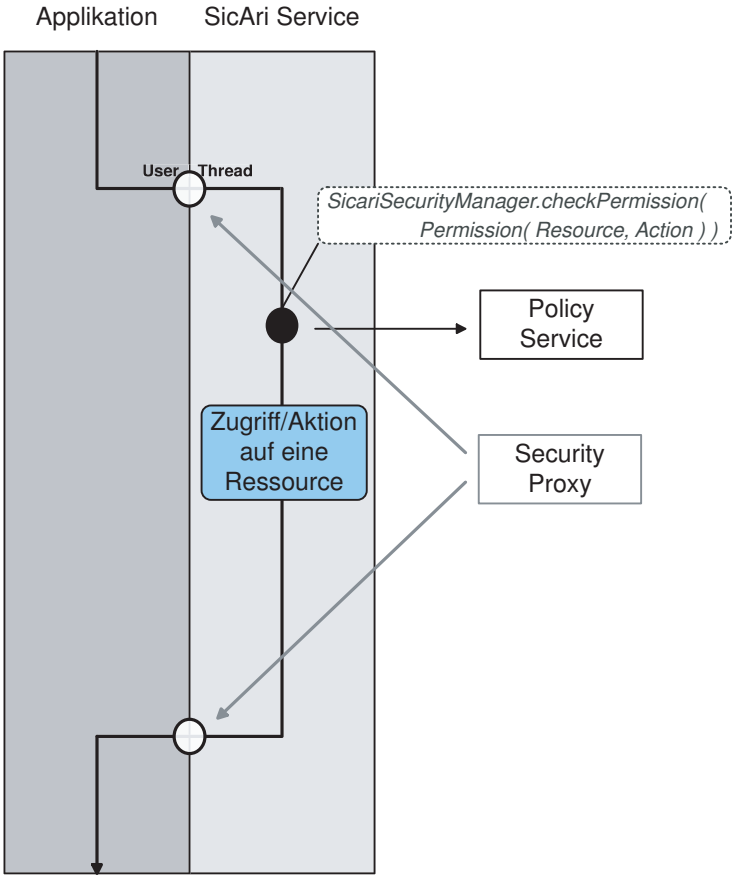

Abbildung 3.3: Zugriffskontrolle und Sessionkontext

[3]Ist in Java für das Laden von Klassen zuständig. Dazu wird der *Bytecode* einer Klasse in die *VirtualMachine* geladen.

[4]Eine `ThreadGroup` bezeichnet eine Gruppe von `Threads`. Mittels eines `ThreadGroup`-Objekts können gleichzeitig mehrere Threads manipuliert werden und gemeinsame Sicherheitsregeln vergeben werden.

Beim Zugriff von einer Applikation auf einen Service wird zunächst beim Übergang von der Applikation zum Service überprüft, ob der anfragende Benutzer ausreichende Rechte besitzt den Service aufzurufen. Ist dies der Fall, so wird der aktuelle Sicherheitskontext temporär gewechselt. Der Dienst wird nun statt im Sicherheitskontext des aktuellen Benutzers im Sicherheitskontext des Benutzers ausgeführt, der den Service ursprünglich publiziert hat.

3.4.2 Sicherheitsarchitektur der SicAri-Plattform

Die SicAri-Sicherheitsarchitektur besteht aus einer Menge von Software- und Hardware-Komponenten, die für den Schutz und die Kontrolle von sicherheitsrelevanten Ressourcen eingesetzt werden. Unter einer Ressource werden innerhalb der SicAri-Infrastruktur beispielsweise Datenbanken, Dienstleistungen wie E-mail oder Drucker-Services sowie schützenswerte Informationen wie Passwörter oder Zertifikate verstanden. Da die Sicherheit in jedem Aspekt des Entwurfs der Plattform eine wichtige Rolle gespielt hat, haben die Sicherheitsfunktionen die Architektur der Plattform maßgeblich beeinflusst.

3.4.2.1 Komponenten des SicAri-Sicherheits-Frameworks

Die SicAri-Plattform definiert für alle sicherheitsrelevanten Aspekte eine Sicherheitsrichtlinie. Eine Sicherheitsrichtlinie umfasst eine Menge von Regeln, die beispielsweise bestimmte Anforderungen an die Vertraulichkeit, die Verschlüsselung von Kommunikationskanälen oder auch Zugriffsrechte festlegen. Alle sensiblen Operationen innerhalb der Plattform werden hinsichtlich ihrer Konformität mit der zugrunde liegenden Sicherheitsrichtlinie geprüft. Zusätzlich können die SicAri-Komponenten die Aspekte der Sicherheitsrichtlinie abfragen, um die passenden Dienste bereitzustellen.
Das Framework zur Etablierung der Sicherheitsrichtlinien besteht aus mehreren Komponenten. Diese sind in Abbildung 3.4 dargestellt. Die Aufteilung in diese Komponenten ist zweckmäßig, um die Mechanismen zur Umsetzung der Richtlinien von der Spezifikation der Richtlinien zu separieren. Eine genaue Beschreibung dieser Komponenten wird in Abschnitt 3.5 vorgestellt.

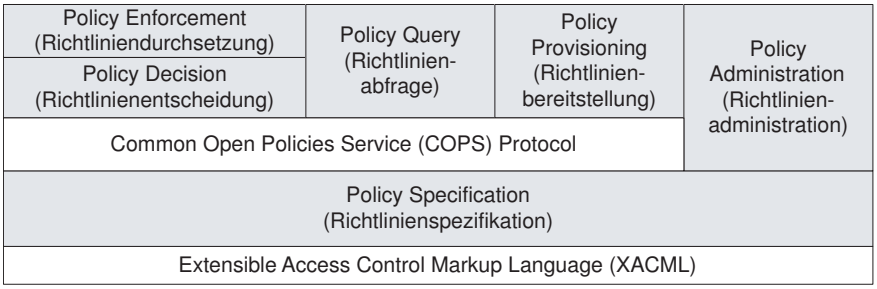

Abbildung 3.4: Komponenten des SicAri-Sicherheits-Frameworks

3.4.2.2 RBAC als Sicherheitsmodell

Das SicAri-Sicherheitsmodell wird durch ein so genanntes *Role-based access control (RBAC)*-Modell [93] umgesetzt. Dieses Modell erlaubt es, Berechtigungen zur Nutzung geschützter Komponenten direkt an Rollen und damit an Aufgaben zu knüpfen. Die Aufgaben werden von Subjekten durchgeführt. Das Sicherheitsmodell legt fest, welche Aufgaben ein Subjekt durchführen darf und damit in welchen Rollen es agieren darf. Das RBAC-System erlaubt es, dass

ein Subjekt gleichzeitig in unterschiedlichen Sitzungen aktiv ist und besitzt damit alle Berechtigungen aller Rollen, in denen es aktiv ist [21]. Weiterführende Informationen sind in [93, 21] zu finden. Wie diese in SicAri umgesetzt werden ist in [74] ausführlich beschrieben.

SicAri erweitert das RBAC-System um zusätzliche Parameter, die bei der Entscheidung über die Gewährung oder Verweigerung einer Berechtigung berücksichtigt werden müssen. Dazu zählt beispielsweise die von einem Benutzer gewählte Authentifikationsmethode. Für den Zugriff auf besonders sensible Dienste der Plattform muss evtl. eine höhere Sicherheit an die Authentifikationsmethode gestellt werden. In diesem Fall reicht ein *Challenge-Response*-Verfahren nicht aus und es wird beispielsweise die Authentifikation über eine *Smart Card* gefordert. Das von SicAri verwendete Modell zur Umsetzung einer Kontext-sensitiven Zugriffskontrolle basiert auf dem Verfahren, das in [124] vorgeschlagen wird.

Bei der Zugriffskontrolle in SicAri sind zahlreiche Komponenten involviert. Neben dem Kontextmanager, der Richtliniendurchsetzungs-, Richtlinienentscheidungs- und Richtlinienspezifikationskomponente können des Weiteren verschiedene Basis- und Anwendungsdienste, die den Kontextmanager mit unterschiedlichen Informationen versorgen, der Plattform beteiligt sein.

3.4.2.3 Referenzmonitor-Modell

SicAri nutzt ein Referenzmonitor-Modell (*Reference Monitor*) [94], auch bekannt als Zugriffsmatrix-Modell, um den Zugriff von Subjekten auf Objekte zu kontrollieren. Dazu wird jeder Zugriff auf Konformität mit der zugrunde liegenden Sicherheitsrichtlinie überprüft. Hierfür werden die Sicherheitsattribute eines Objekts mit denen des Subjekts verglichen.

Der Referenzmonitor besteht aus einer Richtlinienentscheidungskomponente (*Policy Decision Point (PDP)*) und aus ein oder mehreren Richtliniendurchsetzungskomponenten (*Policy Enforcement Point (PEP)*). In der SicAri-Plattform sind diese Komponenten wie bereits erwähnt vollkommen unabhängig von der Richtlinienspezifikationskomponente (*Policy Specification Point (PSP)*). Dadurch kann das Verhalten der Plattform problemlos durch die Spezifikation einer neuen Sicherheitsrichtlinie beeinflusst werden, ohne dass dafür der PEP oder der PDP angepasst werden muss. In Abbildung 3.5 ist der SicAri-Referenzmonitor und angeschlossene Komponenten dargestellt.

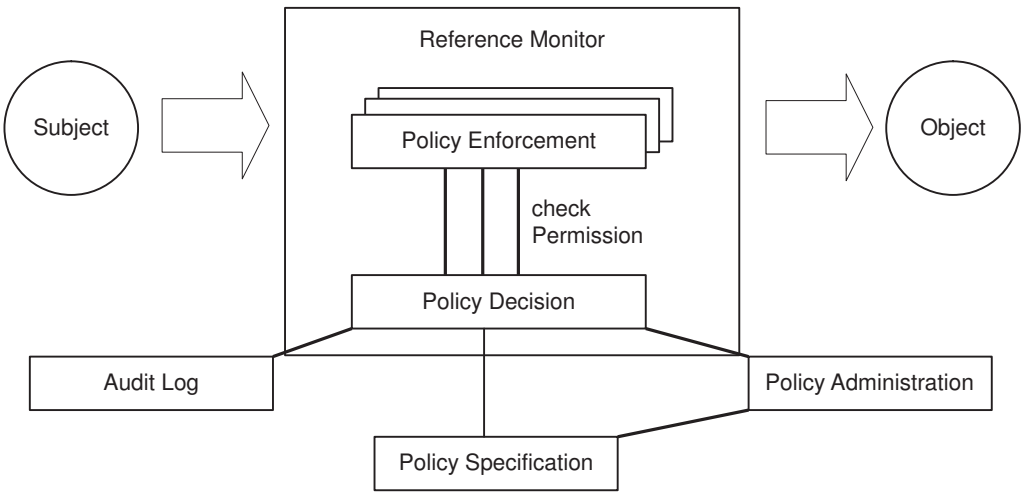

Abbildung 3.5: SicAri Referenz-Monitor

Der Referenz-Monitor wird in der SicAri-Plattform durch den bereits genannten *Security Proxy* realisiert. Er dient als Auslöser für die Richtliniendurchsetzung.
Der Zugriff von Subjekten auf Objekte kann niemals direkt erfolgen. Zuerst überprüft der Referenzmonitor ob der Zugriff gestattet ist. Dazu sendet der PEP eine `checkPermission` Anfrage an den PDP. Der PDP überprüft diese und informiert den PEP über das Ergebnis. Der PEP seinerseits erlaubt oder verweigert nun den Zugriff aufgrund des erhaltenen Ergebnisses auf das Objekt.
In Abbildung 3.5 sind noch drei weitere SicAri-Komponenten dargestellt, die in Beziehung zum Referenzmonitor stehen. Die *Audit Log*-Komponente ist für die Protokollierung aller Ereignisse des PDP zuständig. Dabei werden alle Anfragen und die zugehörigen Ergebnisse protokolliert. Der PSP enthält die Repräsentation der spezifizierten Sicherheitsrichtlinie in einer für das System verarbeitbaren Form. Zur Anpassung dieser Sicherheitsrichtlinie existiert noch die Richtlinienadministrationskomponente *(Policy Administration Point (PAP)*), mit der die Verwaltung der Sicherheitsrichtlinie ermöglicht wird.

3.4.3 Plattform-Kommunikation

Die SicAri-Architektur erlaubt es, die Dienste einer Plattform auch von einer anderen Plattform zu nutzen. Für den Fall, dass ein angeforderter Dienst nicht auf der lokalen Plattform zu finden ist, jedoch auf einer entfernten Plattform existiert, wird eine Kommunikationsverbindung zwischen den beteiligten Plattformen aufgebaut. Der Aufruf eines Dienstes erfolgt deshalb immer in zwei Schritten. Zunächst prüft das *Environment*, ob der Dienst lokal existiert. Ist dies der Fall kann er direkt aufgerufen werden. Wird der Dienst nicht gefunden kommt das SicAri WS-Framework zum Einsatz.
SicAri nutzt wie auch in Abschnitt 3.6.4 beschrieben Web Services um den Zugriff auf entfernte Dienste zu realisieren. SicAri verwendet für die Web Service-Kommunikation die *SOAP-Engine Apache Axis*[5] und zum Auffinden und Veröffentlichen von Web Services die UDDI OpenSource-Implementierung *UDDI4J*[6].
Für die Erstellung von WSDL-Beschreibungen und die Generierung von Java-Klassen aus einer WSDL-Beschreibung werden die Axis-Werkzeuge *Java2WSDL* bzw. *WSDL2Java*[7] verwendet.

Für das weitere Verständnis und die noch folgenden Implementierungen wird deshalb hier ein Überblick über Apache Axis gegeben.

3.4.3.1 SicAri-WS-Framework und Apache Axis

Axis ist ein erweiterbares Framework zur Verarbeitung von SOAP-Nachrichten. Durch Module wird Axis in Teilsysteme untergliedert, wobei die Module jeweils verschiedene Aufgaben übernehmen. Diese Architektur ermöglicht eine flexible Konfiguration von Axis über Konfigurationsdateien.[33]
Axis wird in Form eines Web-Archivs innerhalb einer *Servlet-Engine* (z. B. Tomcat, Jetty, ...) installiert und besteht aus drei *Servlets*[8] (vgl. Abbildung 3.6).

[5] `http://ws.apache.org/axis`
[6] `http://uddi4j.sourceforge.net`
[7] `http://ws.apache.org/axis/java/user-guide.html#UsingWSDLWithAxis`
[8] Ein Servlet ist eine Client/Server-Anwendung, implementiert in der Programmiersprache Java, die es ermöglicht, Client-Anfragen dynamisch zu beantworten. Servlets werden meist zusammen mit HTTP verwendet, um dynamisch HTML-Seiten zu erzeugen und zu verarbeiten.

Abbildung 3.6: Axis im Web Container

Das *Admin-Servlet* dient zur Fernwartung von Axis. Über dieses Servlet können Web Services installiert und deinstalliert werden. Das *Admin-Servlet* ruft intern den Axis-*AdminService* auf. Der AdminService ist ein spezieller Web Service, über den Web Services und *Handler* veröffentlicht und wieder entfernt werden können. Das *SOAP-Monitor-Servlet* ermöglicht es einem Entwickler, die ausgetauschten SOAP-Nachrichten von Web Services zu überwachen bzw. einzusehen. Das *Axis-Servlet* kann die einzelnen Web Services aufnehmen und leitet Web Service-Anfragen an diese weiter.
Die Verarbeitung von SOAP-Nachrichten wird über so genannte Handlerketten abgewickelt. Eine Handlerkette (*Chain*) besteht aus verschiedenen Handler, deren Verkettung anhand von Konfigurationsdateien flexibel angepasst werden können. Typische Aufgaben eines Handler sind beispielsweise die Protokollierung oder die Verarbeitung von Signaturen und Verschlüsselungen einer SOAP-Nachricht.

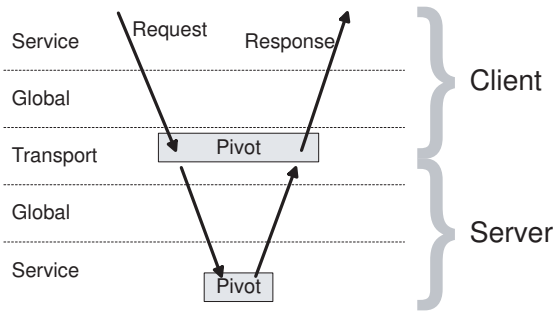

Abbildung 3.7: Axis Pivot-Handler

Die SOAP-Nachrichten durchlaufen in Form eines `MessageContext` die einzelnen Handler. Der `MessageContext` stellt eine Struktur dar, die Referenzen auf die SOAP-Anfrage- und -Antwortnachrichten enthält. Außerdem beinhaltet der `MessageContext` eine Reihe von protokollabhängigen Attributen. Die Verarbeitung von SOAP-Anfragen und SOAP-Antworten

wird von einem *Handler* durchgeführt. Über die `invoke()`-Methode erhält der Handler den `MessageContext` und kann somit auf die Nachrichten zugreifen. Ein Handler, der sowohl eine Anfrage als auch eine Antwort verarbeitet, bzw. eine Anfrage an den dahinterliegenden Service stellt und die Antwort von diesem Service erhält, wird innerhalb von Axis als *Pivot-Handler* bezeichnet. Ein Pivot-Handler ist damit ein spezieller Handler, der einen Wendepunkt in der Nachrichtenverarbeitung darstellt.
In Abbildung 3.7 ist der Weg einer SOAP-Nachricht über die einzelnen Handlerketten vom Client zum Server und wieder zurück zum Client dargestellt. Die Abbildung zeigt die Pivot-Handler auf Client- und Server-Seite, die auf der Serverseite auch als *Provider* und auf der Clientseite als *Transport Sender* bezeichnet werden. Über die Methoden `isClient()` und `getPastPivot()` des `MessageContext` kann festgestellt werden, wo sich eine Nachricht derzeit befindet. Dies ist für die Entwicklung eines Handler sehr hilfreich.
Client und Server bestehen aus einer so genannten Axis-Engine, die die Handlerketten *Transport, Global* und *Service* enthält. Anhand der Transport-Handlerkette können transportspezifische Verarbeitungen einer SOAP-Nachricht vorgenommen werden. Ein Handler in dieser Kette ermöglicht es, spezielle Verarbeitungsschritte in Abhängigkeit des eingesetzten Protokolls, wie zum Beispiel HTTP oder SMTP, durchzuführen. Die globale Handlerkette erlaubt die Web Service-übergreifende Verarbeitung, das heißt dass alle Handler dieser Kette aufgerufen werden, gleichgültig an welchen Web Service eine Nachricht gerichtet ist. Über die servicespezifische Handlerkette können schließlich für einen bestimmten Web Service spezifische Verarbeitungen durchgeführt werden.[123]
Innerhalb der Axis-Engine wird zudem zwischen dem *Request Flow* und dem *Response Flow* unterschieden. Der Request Flow bezeichnet den Weg des `MessageContext`-Objekts hin zum empfangenden Web Service, der Response Flow ist der Rückweg des gleichen Objekts vom Web Service zur anfragenden Instanz. Im Request Flow enthält das `MessageContext`-Objekt zunächst nur eine Referenz auf die Anfragenachricht. Im Response Flow ist dann auch eine Referenz auf die Antwortnachricht enthalten. Handler, die in den Request- bzw. Response Flow eingebettet werden, werden daher auch als Request- bzw. Response-Handler bezeichnet.[123]
Eine spezielle Form von Handlerketten sind die so genannten *Targeted Chains*. Eine Targeted Chain enthält einen Request-Handler für die Verarbeitung des Request Flow, einen Response-Handler für die Verarbeitung des Response Flow und einen Pivot-Handler. Als Targeted Chain werden daher alle transportspezifischen Ketten der Clientseite und alle servicespezifischen Ketten der Serverseite bezeichnet (vgl. Abb. 3.8).[123]

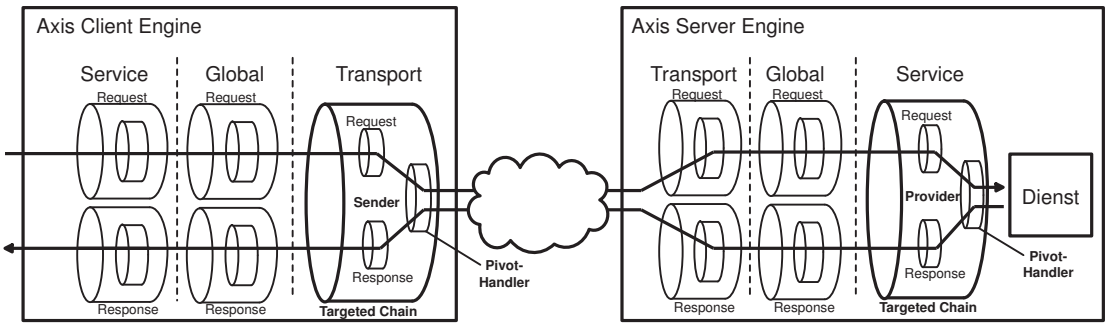

Abbildung 3.8: Axis-Nachrichtenpfad

Abbildung 3.8 zeigt den Weg einer Nachricht über die einzelnen Handlerketten auf Client- und Server-Seite. Auf Client-Seite stellt beispielsweise ein Benutzer eine Anfrage. Diese Anfrage

wird von Axis entgegengenommen und in einen `MessageContext` eingebettet. Die Anfrage durchläuft nun in Form des `MessageContext` die einzelnen Handlerketten und wird schließlich über ein Transportprotokoll versendet. Auf der Server-Seite nimmt ein Handler der Transportkette die Nachricht entgegen und packt diese wieder in einen `MessageContext`. Der `MessageContext` durchläuft nun die einzelnen Handlerketten bis zum eigentlichen Dienst. Nach dem der Dienst die Anfrage bearbeitet hat, kehrt sich der Nachrichtenfluss um. Der erstellte `MessageContext` enthält jetzt zusätzlich eine Referenz auf die Antwort und durchläuft nun die einzelnen Handlerketten in umgekehrter Reihenfolge bis zum Client der die Anfrage initiiert hat.[33]

3.5 Basisdienste der SicAri-Plattform

3.5.1 Security Policy Architektur

Die SicAri *Security Policy*-Architektur besteht aus zahlreichen Komponenten, die im Folgenden vorgestellt werden. Abbildung 3.9 stellt diese Komponenten und deren Interaktion dar.

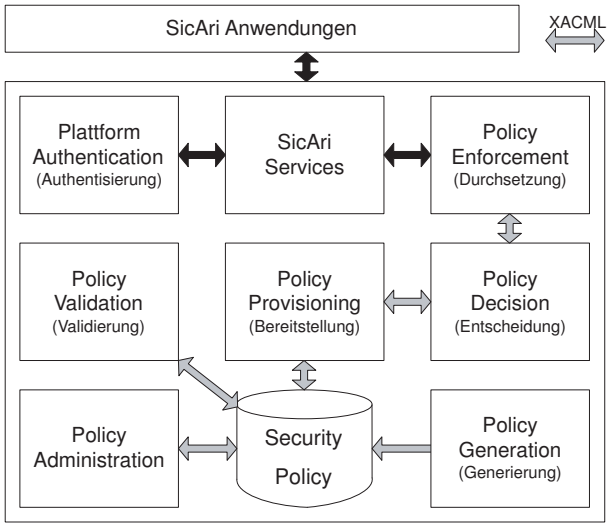

Abbildung 3.9: Komponenten der SicAri Security-Policy Architektur

- **Policy Generierung** Über die *Policy Generation*-Komponente werden halbautomatisch Policies generiert. Dazu wählt der Administrator ein Policy-Muster (*Policy-Pattern*) aus. Danach werden die Benutzer der Plattform den verschiedenen Rollen des Policy-Pattern zugeordnet. Anhand dieser Angaben kann dann automatisiert eine XACML-Policy erstellt werden.

- **Policy Validierung** Über die *Policy Validation*-Komponente werden die Sicherheitsrichtlinien hinsichtlich der gestellten Sicherheitsziele überprüft. Hierbei wird auch das Zusammenspiel der Sicherheitsregeln kontrolliert. Damit kann beispielsweise festgestellt werden, ob sich Regeln evtl. gegenseitig aushebeln und so Sicherheitslücken im System entstehen können.

- **Policy Administration** Die *Policy Administration*-Komponente ermöglicht es Sicherheitsrichtlinien zu administrieren. Dies kann beispielsweise notwendig sein, wenn neue Benutzer dem System hinzugefügt werden oder die Rechte eines Benutzers angepasst werden müssen.

- **Policy Bereitstellung** Die *Policy Provisioning*-Komponente ist für die Bereitstellung und Verteilung der Sicherheitsrichtlinien zuständig. Werden Sicherheitsrichtlinien verändert, so werden diese Änderungen an die angeschlossenen Komponenten automatisch weitergegeben, so dass die *Policy Decision*-Komponente ihre Entscheidungen immer anhand der aktuellsten Sicherheitsrichtlinie treffen kann.

- **Policy Entscheidung** Alle Entscheidungen bezüglich der Sicherheitsrichtlinie werden durch die *Policy Decision*-Komponente getroffen. Die in SicAri verwendete *Policy Decision*-Komponente basiert auf einer erweiterten Version der XACML-Implementierung[9] von SUN Microsystems.[80]

- **Policy Durchsetzung** Die *Policy Enforcement*-Komponente garantiert, dass alle sicherheitsrelevanten Aktivitäten nur dann ausgeführt werden, wenn sie mit der zugrunde liegenden Sicherheitsrichtlinie übereinstimmen. Die Komponente erkennt dazu alle sicherheitsrelevanten Aktivitäten und fragt bei der *Policy Decision*-Komponente nach, ob die Aktivität ausgeführt werden darf oder nicht.

- **SicAri Services** In SicAri wird die Interaktion zwischen den Anwendungen und den Services und den Services untereinander sowie der Zugriff von Services auf lokale Ressourcen implizit durch die *Policy Enforcement*-Komponente kontrolliert. Die Verarbeitung der Sicherheitsrichtlinie wird für den Service transparent durchgeführt, das heißt ein Service muss nicht zwingend wissen, welche Sicherheitsrichtlinie im Hintergrund existiert.

- **Plattform Authentifikation** Für die Durchsetzung der Sicherheitsrichtlinien ist die erfolgreiche Authentifikation aller Entitäten der SicAri-Infrastruktur eine zwingende Voraussetzung. In SicAri existieren mehrere Module, die die Authentifikation der Benutzer bei unterschiedlichen Anmeldeszenarien (siehe Kapitel 3.5.3) übernehmen.

3.5.2 Kontextmanager

Der *Kontextmanager* stellt für die Durchsetzung der Sicherheitsrichtlinie verschiedene Informationen über das Environment und der darin agierenden Entitäten wie beispielsweise Benutzer, Dienste oder Geräte zur Verfügung. Der Kontextmanager speichert und stellt diese Informationen für andere Benutzer und Dienste zur Verfügung und wird daher durch eine systemweite Zugriffskontrollrichtlinie kontrolliert. Weitere Details zum Kontextmanager sind [74] zu entnehmen.

3.5.3 Authentifikationsmanagement

Die SicAri-Plattform stellt die Möglichkeit des SSO durch eine Reihe von unterschiedlichen Authentifikationsmechanismen zur Verfügung. Beim Login wird die Identität eines Benutzers einmal festgestellt und daraufhin eine Benutzersitzung (*Session*) aufgebaut. Eine Session wird durch einen Sicherheitskontext (*Security Context*) repräsentiert. Dieser Kontext stellt die Informationen, die für die Durchsetzung der Sicherheitsrichtlinien benötigt werden, bereit. Solange

[9] http://sunxacml.sourceforge.net

ein gültiger Sicherheitskontext für einen Benutzer existiert, kann er alle Dienste der Plattform benutzen ohne sich dafür erneut authentifizieren zu müssen. Erst nach einem Logout oder dem Ablauf der Gültigkeit des Sicherheitskontext muss sich der Benutzer erneut authentifizieren.[80]
Die Plattform stellt Mechanismen für die lokale und entfernte Authentifikation zur Verfügung. Ist der globale *Authentifikationsmanager (AM)* erreichbar so wird er dazu verwendet, die Authentifikation mittels eines Challenge-Response Protokolls über den lokalen AM durchzuführen. Außerdem erstellt der AM ein *Security Token (ST)* für den Benutzer. Dieses Security Token ist mit dem privaten Schlüssel des AM signiert. Es wird auf der lokalen Plattform, an der sich der Benutzer angemeldet hat, gespeichert und wird mit seinem Sicherheitskontext verknüpft. Ruft ein Benutzer einen Dienst einer entfernten Plattform auf, wird das Security Token implizit an diese Plattform übertragen und authentifiziert so den Benutzer gegenüber dieser Plattform.
Ist der globale AM nicht verfügbar, so können sich nur Benutzer anmelden, deren Sicherheitsinformationen der lokalen Plattform zur Verfügung stehen. Diese Benutzer müssen auf der lokalen Plattform vorkonfiguriert sein. Nur durch diesen Mechanismus kann der Plattform-Administrator ohne bestehende Verbindung zum AM die Plattform starten. Bei der lokalen Authentifikation wird das Security Token des Benutzers mit dem privaten Schlüssel des Plattform-Administrators signiert. Will ein Benutzer später auf einen Dienst einer entfernten Plattform zugreifen, muss er erst über den globalen AM authentifiziert werden.
In Abbildung 3.10 ist das Zusammenspiel der einzelnen Komponenten des Authentifikationsmechanismus dargestellt.

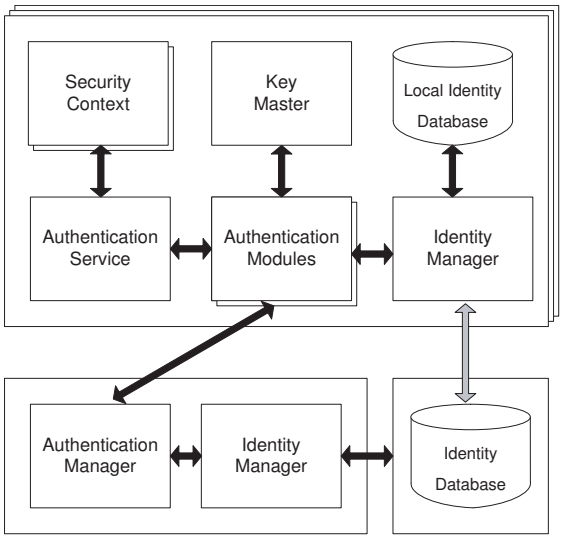

Abbildung 3.10: Komponenten des SicAri-Authentifikationsmanagement

Ein *Security Token* setzt sich aus den folgenden Attributen zusammen:

- `version` Gibt die Version des Security Token an.
- `tokenID` Diese ID identifiziert das Security Token Plattform-übergreifend. Die ID setzt sich aus der Benutzer-ID und der Sitzung-ID zusammen.
- `creationTime` Der Zeitpunkt zu dem das Token erstellt wurde.
- `validSince` Ein Zeitpunkt, der definiert ab wann das Token gültig ist.

- `validTill` Ein Zeitpunkt, der definiert bis wann das Token gültig ist.
- `attributes` Hier können optionale Attribute hinterlegt werden. In einem Attribut wird beispielsweise die verwendete Authentifikationsmethode (z. B. Smart Card, JKS, Passwort) gespeichert.
- `issuerCertificate` Das X.509 Zertifikat des Token-Ausstellers.
- `signature` Alle anderen Werte des *Security Token* werden mit dem privaten Schlüssel des globalen Authentifikationsmanagers bzw. des Plattform-Administrators signiert. Der daraus berechnete Signaturwert wird in diesem Attribut abgelegt.

Für die Übertragung eines Security Token wird dieses in verschiedenen Formaten kodiert. Die Plattform unterstützt die Formate ASN.1, SAML (siehe Kapitel 2.5.4) sowie ein proprietäres XML-Format.

- **ASN.1** Die *Abstract Syntax Notation One (ASN.1)* kodiert das Security Token in einem binären Format. ASN.1 ist eine Beschreibungssprache die als Standard bei der *International Telecommunication Union (ITU)*[10] veröffentlicht ist.
- **SAML** Für die Nutzung von Security Tokens innerhalb der Web Service-Welt bietet sich eine Kodierung mittels SAML an. Innerhalb von SicAri wurde zu diesem Zweck das SAML *Attribute Assertion* entsprechend erweitert.
- **PlainXML** Um auch andere auf XML basierenden Protokolle zu unterstützen, die evtl. das SAML-Format nicht verstehen, wurde noch ein proprietäres XML-Format eingeführt. Für die Verschlüsselung und Signatur werden hier die Standards XMLdsig und XMLenc verwendet.

3.5.4 Identity Management

Das *Identity Management* umfasst die Administration von Identitäten in der SicAri-Plattform sowie die Bereitstellung des *Identity Manager*-Dienstes. Abbildung 3.11 stellt die involvierten Komponenten des Identitätsmanagement dar.

3.5.4.1 Identity Administration

Unter der *Identity Administration* ist die Registrierung von neuen Identitäten, die Modifikation von Identitätsattributen sowie das Löschen von Identitäten zu verstehen. Alle Identitäten und deren Attribute werden innerhalb der SicAri-Infrastruktur in einer globalen Identitätsdatenbank gespeichert und administriert.
Die SicAri-Plattform unterscheidet generell drei Typen von Identitäten bzw. Benutzern.

- **Plattform Administrator** Jede Plattform-Instanz in einer SicAri-Infrastruktur wird von einem so genannten Plattform-Administrator gestartet. Die kryptografischen Schlüssel des Plattform-Administrators werden dazu verwendet, die einzelnen Plattform-Instanzen innerhalb der SicAri-Infrastruktur eindeutig zu identifizieren.

[10] `http://www.itu.int`

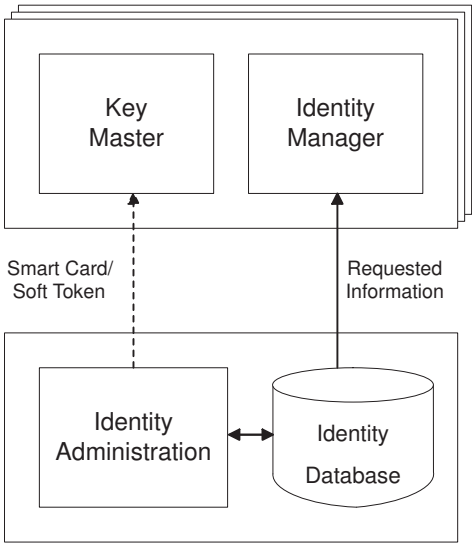

Abbildung 3.11: Komponenten des SicAri-Identitätsmanagement

- **Service User** Die Basisdienste der Plattform werden durch so genannte *Service User* initialisiert bzw. gestartet. Dies sind nur abstrakte Benutzer, die in der Regel keine echten Personen darstellen. Der Service User wird lediglich für die Vergabe von gesonderten Rechten für die einzelnen Basisdienste benötigt.
- **Regular User** Ein regulärer Benutzer ist in der Regel ein Benutzer, der die Dienste einer Plattform verwendet. In Abhängigkeit seiner Rechte ist er entweder auf die reine Nutzung der Dienste beschränkt oder kann auch die Administration von bereitgestellten Diensten übernehmen.

Eine Identität besitzt verschiedene Attribute, die in Abhängigkeit der benötigten Funktionen auch erweitert werden können. Generell besitzt eine Identität die folgenden Attribute:

- **userid** Die *userid* wird benötigt, um einen Benutzer eindeutig innerhalb der SicAri-Infrastruktur zu identifizieren.
- **dname** Dieses Attribut wird dazu verwendet einen Benutzer in einem hierarchisch aufgebauten Namensraum eindeutig zu identifizieren. Dieser Wert muss mit dem `Subject`-Feld des verwendeten X.509 Zertifikats übereinstimmen.
- **userpassword** Das Benutzerpasswort kann zusammen mit der Benutzer-ID verwendet werden, um einen Benutzer am System zu authentifizieren. Das Benutzerpasswort wird nicht direkt abgespeichert, sondern nur der davon abgeleitete Hash-Wert.
- **usercertificate** Dieses Attribut ermöglicht es bis zu drei X.509 Zertifikate anzugeben. Diese werden jeweils für die Bereiche Authentifikation, Verschlüsselung und Signatur verwendet.

3.5.4.2 Identity Manager

Der *Identity Manager* ist ein Basisdienst der Plattform. Er stellt für die einzelnen Dienste der Plattform verschiedene Funktionen zur Verfügung. Anhand von Benutzerattributen können ein-

zelne Benutzer gesucht werden. Dazu liefert der Identity Manager eine Liste von Benutzer-IDs zurück. Anhand von Benutzer-IDs können außerdem die einzelnen Attribute eines Benutzers abgefragt werden. Er existiert lokal in jeder SicAri-Plattform und besitzt eine Verbindung mit der globalen Identitätsdatenbank.
Über den Identity Manager wird der Zugriff auf die einzelnen Zertifikate eines Benutzers ermöglicht. Über die öffentlichen Schlüssel können dann die kryptografischen Funktionen ausgeführt werden, die vom *Key Master* bzw. *Key Manager* bereitgestellt werden.

3.5.5 Key Manager

Der *Key Manager* bzw. Schlüsselmanager ist eine lokale Komponente der SicAri-Plattform und besitzt die folgenden Funktionen:

- Speicherung der chiffrierten privaten Schlüssel:
  - Interner Schlüsselspeicher:
    Der interne Schlüsselspeicher verwendet die Klasse `KeyStore`, um die privaten Schlüssel abzuspeichern. Diese werden in einer lokalen Datei Passwort-verschlüsselt abgespeichert. Für den SSO-Mechanismus wird nach jeder Authentifikation der private Schlüssel des Benutzers zwischengespeichert.
  - Externe Schlüsselspeicher:
    Der private Schlüssel befindet sich ausschließlich in einem externen Schlüsselspeicher, wie einer Smart Card oder einem *Schlüsseltoken*. Der Key Manager puffert in diesem Fall nur das zugehörige Passwort bzw. die PIN.
- Verbindung und Interaktion mit einem externen Schlüsselspeicher wie beispielsweise einer Smart Card. Die Verbindung wird in diesem Fall über eine entsprechende Bibliothek des Herstellers der Smart Card hergestellt. Der private Schlüssel verlässt niemals die Smart Card.
- Zwischenspeicherung von Schlüsseln zur Umsetzung des SSO-Mechanismus für den Authentifikationsmanager. Wie bereits erwähnt wird entweder der Schlüssel oder eine PIN bzw. ein Passwort für eine festgelegte Zeit zwischengespeichert.
- Bereitstellung von Funktionen für den Authentifikationsmanager, um Signaturen oder asymmetrische Verschlüsselung zu erstellen. Diese Funktionen werden direkt im Key Manager durchgeführt.

Der Key Manager verwaltet also die öffentlichen und privaten Schlüssel der authentisierten Benutzer bzw. die Referenzen auf die Schlüssel (wenn eine SmartCard verwendet wird). Der Key Manager stellt außerdem sicher, dass ein Benutzer nur Zugriff auf seinen privaten Schlüssel erhält. Auf alle anderen privaten Schlüssel kann der Benutzer nicht zugreifen. Innerhalb von SicAri wird der Key Manager durch ein Objekt der Klasse `KeyMaster` repräsentiert.

3.5.6 Persistency Service

Die SicAri-Plattform stellt einen Mechanismus zur Verfügung, der es ermöglicht Daten abzuspeichern. Der „*Persitency Service*“ stellt eine Schnittstelle zum Speichern und Abrufen von beliebigen Daten zur Verfügung. Hierbei beachtet der Service auch die zugrunde liegende Sicherheitsrichtlinie und garantiert somit, dass Komponenten nur Zugriff auf Daten bekommen, für die sie auch entsprechend autorisiert sind.

3.6 Anwendungsdienste der SicAri-Plattform

Im Folgenden wird ein kurzer Überblick über die existierenden Anwendungsdienste der Plattform gegeben.

3.6.1 Sensormodule

In der SicAri-Plattform existieren verschiedene Sensoren, die Daten der Umgebung an den SicAri-Kontextmanager (vgl. Kapitel 3.5.2) liefern. SicAri unterscheidet dazu Sensoren, die beispielsweise Standort-Informationen oder das aktuelle Datum und die Uhrzeit liefern. Diese Sensoren liefern Daten aus der realen Welt, im Gegensatz zu den Sensoren, die verschiedene Daten aus der SicAri-Infrastruktur liefern. Hier kommen beispielsweise Ereignisse in Betracht, die entstehen, wenn ein Benutzer bestimmte Ressourcen aufruft. Ein so genanntes *Sensor-Modul* erweitert den Kontext-Manager und stellt ihm für verschiedene Aufgaben notwendige Informationen bereit. Es werden zwei Arten von Sensor-Modulen unterschieden.
Aktive Sensor-Module sorgen automatisch dafür, dass die entsprechenden Daten im Kontext-Manager aktualisiert werden. Dies wird in bestimmten Intervallen vorgenommen, die abhängig von den Daten festgelegt werden. Diese Art der Sensoren werden über Ereignisse aktiviert oder werden als Thread implementiert.
Passive Sensor-Module sind immer mit einem bestimmten Knoten in der Kontext-Manager-Hierarchie verbunden und werden durch den Kontext-Manager aufgerufen, wenn der Inhalt dieses Knotens von einem anderem Dienst benötigt wird. Wenn ein Sensor-Modul vom Kontext-Manager aufgerufen wird, so ermittelt das Sensor-Modul die notwendigen Daten und liefert sie direkt zurück. Passive Sensor-Module müssen das Interface `SensorModule` des *Java-Package* `de.sicari.context` implementieren.

3.6.2 Kryptografische Funktionen

Die Sicherheit derzeitiger kryptografischer Verfahren kann nicht für die komplette Zukunft garantiert werden. Durch technische Neuerungen oder neue Erkenntnisse in der Kryptoanalyse können diese Verfahren evtl. gebrochen werden und daher die Sicherheit nicht mehr garantieren. Anwendungen, die diese kryptografischen Verfahren als Grundlage zur Umsetzung ihrer Sicherheitsziele verwenden, müssen daher evtl. überarbeitet werden. Zu diesem Zweck ist es sinnvoll, ein so genanntes Provider-Modell einzusetzen, das die kryptografischen Verfahren von der Anwendung entkoppelt und so den Austausch von evtl. unsicher gewordenen Verfahren ohne großen Aufwand ermöglicht.
Die SicAri-Plattform verwendet hierfür das Provider-Modell, das bereits durch den Einsatz der Java-Technologie zur Verfügung steht. Für die Bereitstellung von Signatur-Algorithmen existiert hierfür die *Java Cryptography Architecture (JCA)*. Die *Java Cryptography Extension (JCE)* erweitert dieses Modell um Verschlüsselungsverfahren. Beide Konzepte stellen allgemein anerkannte kryptografische Algorithmen bereit. Außerdem ist es möglich, eigene Entwicklungen in diese Konzepte einzubinden. Dies geschieht durch die Implementierung entsprechend vordefinierter Schnittstellen.

3.6.3 Kommunikationsprotokolle

SicAri schränkt die Nutzung von Kommunikationsprotokollen nicht ein. Die verschiedenen Dienste der Plattform können sowohl standardisierte als auch proprietäre Kommunikationsprotokolle nutzen, wenn diese die Sicherheitsziele nicht verletzen. Dazu schlägt die SicAri-

Arbeitsgruppe *Protocol Engineering* entsprechende Protokolle für die unterschiedlichen Kommunikationsanforderungen vor. Die Arbeitsgruppe evaluiert hierfür die Protokolle und überprüft, ob sie die gestellten Sicherheitsziele nicht verletzen.

3.6.4 Web Services

Ein wichtiger Aspekt des SicAri-Kernels ist das *Service Management* und *Service Discovery* (ermöglicht das Auffinden und Aufrufen von Services). Lokal wird dies vom *Environment* übernommen. In einer verteilten Umgebung braucht es jedoch einen anderen Mechanismus, um das Auffinden und Aufrufen von Services einer entfernten SicAri-Plattform zu ermöglichen. Wie in Kapitel 3.4.3 beschrieben wird dafür ein Namensdienst (*Naming Service*) und ein dediziertes Kommunikationsprotokoll benötigt.
Die SicAri-Spezifikation „Interoperability with Component Standards and Web Services“ [73] schlägt dazu die Implementierung einer zusätzlichen Schicht auf Basis von Web Services vor. Diese ermöglicht das *Service Management* und das *Service Discovery* in einer verteilten Umgebung und fördert zudem die Interoperabilität.
Im Kontext von SicAri werden daher die drei Standards UDDI, WSDL und SOAP der Web Services-Architektur mit Hilfe von existierenden *Open-Source* Implementierungen umgesetzt.
Das Ziel der Spezifikation ist die Erweiterung des lokalen Servicemanagers in transparenter Weise für den Benutzer. Dienste, die im lokalen *Environment* veröffentlicht werden, werden zusätzlich als Web Services für einen entfernten Zugriff bereitgestellt. Dazu wird automatisch eine WSDL-Beschreibung generiert und diese im UDDI-Verzeichnis abgelegt. Ein Zugriff auf einen Dienst, der nicht in der lokalen Plattform existiert, wird dann automatisch über einen Web Service zum entfernten Dienst umgeleitet. Für weitere Informationen sei auf die Spezifikation [73] verwiesen.

Kapitel 4

Realisierung

In diesem Kapitel werden die im Rahmen dieser Arbeit vorgenommenen Implementierungen vorgestellt. Dazu werden diese zunächst auf einer mehr abstrakten Ebene konzipiert. Danach wird die Implementierung im Detail vorgestellt. Schließlich wird noch auf die Integration der implementierten Komponenten in die SicAri-Plattform eingegangen.

4.1 Konzeption

4.1.1 Authentifikation

Bei der Authentifikation von Benutzern innerhalb der SicAri-Infrastruktur sind verschiedene Komponenten beteiligt. Diese sind in Abbildung 4.1 dargestellt. Der Benutzer meldet sich entweder direkt an einer lokalen SicAri-*Plattform (PF)* an oder identifiziert sich über ein anderes Endgerät bei der Plattform. Das lokale JAAS-Framework (siehe nächstes Kapitel) realisiert in Zusammenarbeit mit dem Authentifikationsservice die Authentifikation des Benutzers. Der Identitätsmanager (vgl. Kapitel 3.5.4.2) prüft dazu die Identität des Benutzers in seiner Datenbank. Der Schlüsselmanager (vgl. Kapitel 3.5.5) verwaltet die öffentlichen und privaten Schlüssel der Benutzer in sicherer Weise.
Besitzt die lokale Plattform eine Verbindung zu dem globalen *Authentifikationsmanager (AM)* (vgl. Kapitel 3.5.3), wird die Authentifikation des Benutzers mittels eines auf Web Services basierenden Protokolls an diesen Dienst delegiert. Der AM überprüft die Identität durch Abfrage dieser über den Identitätsmanager, der seinerseits die globale Identitätsdatenbank konsultiert. Bei der Authentifikation wird zudem über den Schlüsselmanager auf die öffentlichen Schlüssel eines Benutzers zugegriffen. Die Verbindung zwischen einem Endgerät des Benutzers und der lokalen PF, der lokalen PF und dem AM sowie der Kommunikation mit der UDDI-Registry werden ggf. durch entsprechende Maßnahmen wie Verschlüsselung und Signatur geschützt.

4.1.1.1 JAAS

SicAri nutzt für die Authentifikation von Benutzern das *Java Authentication and Authorization Service (JAAS)*-Framework. Dieses Framework ermöglicht die Umsetzung von Zugriffsrechten und die Bereitstellung von Diensten zur Authentifikation innerhalb von Java-Programmen.
Für die weitere Betrachtung ist es daher notwendig, einen kurzen Überblick über das JAAS-Framework zu geben. Die Abbildung 4.2 verdeutlicht anhand eines Sequenzdiagramms den Ablauf und die beteiligten Komponenten einer Benutzerauthentifikation. Für die Authentifikation

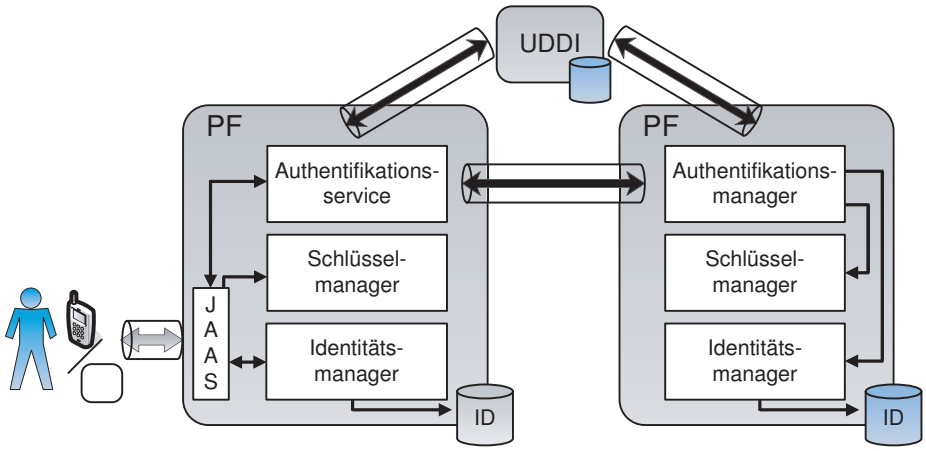

Abbildung 4.1: Beteiligte Komponenten der Benutzerauthentifikation

wird zunächst ein so genannter `LoginContext` erstellt. Dieser wird mit einem Profil, einem `CallbackHandler` sowie der `Configuration` initialisiert. Der `CallbackHandler` dient zur Abfrage von Benutzernamen und Passwörtern oder anderen Informationen von dem anzumeldenden Benutzer. Die `Configuration` stellt eine Konfigurationsdatei dar, die eine Reihe von so genannten *Login-Module* (`LoginModule`) enthält. Die Reihenfolge dieser Login-Module sowie verschiedene Attribute definieren die Reihenfolge, in der die einzelnen Login-Module innerhalb des JAAS-Frameworks zur Authentifikation von Benutzern verwendet werden. Die Konfigurationsdatei gruppiert darüber hinaus Login-Module in verschiedene Profile. Welches Profil zur Authentifikation verwendet werden soll, wird bei der Initialisierung des `LoginContext` festgelegt. Diese Steuerung des Authentifikationsprozesses anhand einer Konfigurationsdatei ist äußerst flexibel und ermöglicht es, nachträglich die Authentifikation zu erweitern, ohne dass hierfür Änderungen an der bestehenden Software vorgenommen werden müssen. Dieses Konzept ist eine Implementierung des so genannten *Pluggable Authentication Module (PAM)*-Framework (vgl. [67]).

Nach der Initialisierung erstellt der `LoginContext` ein `Subject` und ruft danach die einzelnen Login-Module anhand ihrer definierten Reihenfolge auf. Innerhalb der einzelnen Login-Module werden verschiedene Login-Verfahren umgesetzt. Hier wird ein Benutzer beispielsweise per Smart Card oder Passwort authentisiert. Konnte ein Benutzer mit einem Login-Modul erfolgreich authentisiert werden, werden so genannte `Principal`-Objekte erstellt und diese dem `Subject` hinzugefügt. Andernfalls wird das nächste Login-Modul aufgerufen.

Als `Prinicipal` wird innerhalb des JAAS-Framework eine Entität wie beispielsweise ein Benutzername oder eine Login-ID bezeichnet. Ein `Principal` wird zur Identifikation von Benutzern verwendet.

Zudem können dem `Subject` innerhalb des Login-Modul weitere so genannte *Credentials* hinzugefügt werden. Als Credential werden sicherheitsrelevante Attribute wie beispielsweise die öffentlichen und privaten Schlüssel eines Benutzers bezeichnet. Ein Credential kann innerhalb des JAAS-Framework eine beliebiges Objekt darstellen. Der Zugriff auf die Credentials wird durch Zugriffsrechte reglementiert.

Die Credentials werden zudem in öffentliche und private Credentials unterschieden. Die privaten Credentials sind vor Zugriffen durch andere Benutzer geschützt. Auf die öffentlichen Credentials kann normalerweise ohne Beschränkung zugegriffen werden.

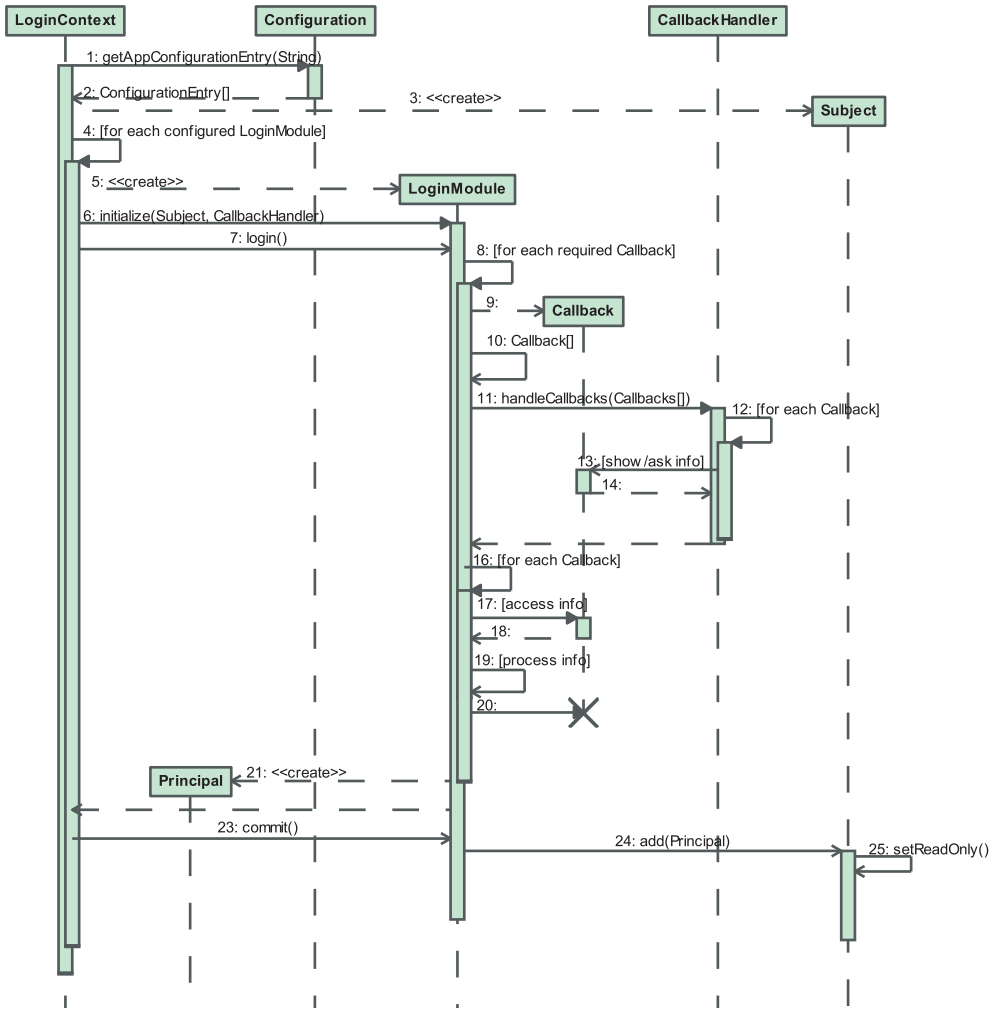

Abbildung 4.2: Sequenzdiagramm: Benutzerauthentifikation mit dem JAAS-Framework

Sind alle Credentials und `Principal`-Objekte dem `Subject` hinzugefügt worden, wird das `Subject` in der Regel mittels der Methode `setReadOnly()` vor allen weiteren schreibenden Zugriffen geschützt. Damit ist der Authentifikationsprozess abgeschlossen. Auf Basis des erstellten `Subject` können nun weitere Autorisierungsentscheidungen getroffen werden.

4.1.1.2 Security Token und Soft Token

Für die weiteren Betrachtungen der Benutzerauthentifikation ist es notwendig die evtl. missverständlichen Begriffe *Security Token* und *Soft Token* klar voneinander abzugrenzen:

Definition 4.1 *Security Token: Innerhalb von SicAri bezeichnet ein Security Token (ST) ein Objekt mit verschiedenen Attributen, die Sicherheitsinformationen repräsentieren. Ein ST kann unterschiedlich kodiert sein. Die SicAri-Plattform unterstützt derzeit STs im SAML-, ASN.1 und XML-Format. Ein ST ist das Ergebnis einer erfolgreichen Authentifikation und wird von einem speziellen Dienst ausgestellt (vgl. Kapitel 3.5.3).*

Definition 4.2 *Soft Token: Als Soft Token wird eine Datenstruktur in Form einer Datei verstanden, die als Schlüsselspeicher (Key Store) dient. Sie beinhaltet die privaten Schlüssel bzw. Zertifikate eines asymmetrischen Kryptosystems, die zur Authentifikation von Benutzern verwendet werden können.*

4.1.1.3 Authentifikationsmethoden & -ebenen

Authentifikationsmethoden Wie bereits in Kapitel 2.1.2.1 beschrieben, existieren generell drei verschiedene Authentifikationsmethoden. Innerhalb von SicAri kommen diese Verfahren einzeln oder auch in Kombination zur Anwendung. Die Authentifikation mittels biometrischer Merkmale wird zurzeit noch nicht unterstützt.

- $T1$: Wissensbasierte Authentifikation, beispielsweise durch Benutzername/Passwort
- $T2$: Besitzbasierte Authentifikation, durch einen Gegenstand wie z. B. eine Smart Card
- $T3$: Besitzbasierte Authentifikation, durch biometrische Merkmale (Iris-Scan, Stimmerkennung, ...)

Innerhalb von SicAri werden derzeit vier Ausprägungen dieser Verfahren unterstützt. Welches Verfahren für einen Benutzer jeweils zur Verfügung steht, ist davon abhängig, welche Informationen (Passwort, Zertifikate, Smart Card, ...) durch den Identitätsadministrator für den jeweiligen Benutzer angelegt wurden:

- $M1$: Authentifikation mit Benutzername und Passwort. Dazu wird beim Anlegen des Benutzers der zugehörige Passwort-Hash-Wert in der Identitätsdatenbank mit abgespeichert. Anhand dieses Hash-Werts kann der Benutzer später authentifiziert werden. (Authentifikationstyp $T1$).
- $M2$: Authentifikation mittels Soft Token. Beim Anlegen des Benutzers werden öffentliche und private Schlüssel erzeugt. Der öffentliche Schlüssel wird in der Identitätsdatenbank abgespeichert. Der private Schlüssel wird mittels Passwort geschützt und dem Benutzer in Form eines Key Store zur Verfügung gestellt. (Authentifikationstyp $T2$ in Kombination mit Typ $T1$). Unter diesem Verfahren kann auch das SSL/TLS-Verfahren verstanden werden, wenn hier die Option der Client-Authentifikation aktiviert wird und beim Verbindungsaufbau das in dem Soft Token gespeicherte Zertifikat verwendet wird.
- $M3$: Authentifikation mittels Smard Card. Beim Anlegen des Benutzers werden öffentliche und private Schlüssel erzeugt. Der öffentliche Schlüssel wird in der Identitätsdatenbank abgespeichert. Der private Schlüssel wird auf der Smart Card des Benutzers gespeichert und ist dort mittels einer PIN vor unberechtigten Zugriffen geschützt. (Authentifikationstyp $T2$ in Kombination mit Typ $T1$). Unter diesem Verfahren kann auch das SSL/TLS-Verfahren verstanden werden, wenn hier die Option der Client-Authentifikation aktiviert wird und beim Verbindungsaufbau das auf der Smart Card gespeicherte Zertifikat verwendet wird.
- $M4$: Authentifikation mittels ST. Hierfür muss zunächst eine ST von einer vertrauenswürdigen Instanz in der SicAri-Infrastruktur ausgestellt worden sein. Bei der Authentifikation mittels ST wird sowohl die Gültigkeit also auch die Vertrauenswürdigkeit der ausstellenden Instanz überprüft. (Authentifikationstyp $T2$).

Authentifikationsebenen Die SicAri-Plattform ermöglicht die Authentifikation von Benutzern auf verschiedenen Ebenen bzw. über verschiedene Wege (vgl. [80]). Diese können dabei unterschiedliche Sicherheitsstufen reflektieren. Mit der Sicherheitsstufe sind wiederum differenzierte Zugriffsregeln verbunden. Daher wird die jeweilige Authentifikationsmethode als ein Attribut in dem, nach erfolgreicher Authentifikation, erstellten ST gespeichert. Die PF unterstützt die Authentifikation über folgende Ebenen.

- $L1$: Benutzer-Login über die SicAri-Shell. Hierbei sind die Authentifikationsmethoden $M1$-$M4$ möglich.
- $L2$: Ein Anwendung fordert einen Benutzer-Login über die SicAri-Authentifikations-API. Hierbei sind die Authentifikationsmethoden $M1$-$M4$ möglich.
- $L3$: Der Administrator personifiziert einen Service-Benutzer (vgl. Kapitel 3.5.4.1). Hierbei muss der Administrator bereits mittels $M1$-$M4$ authentifiziert sein. Die Personifizierung erfolgt dann über das Verfahren $M4$.
- $L4$: Authentifikation über die SicAri-Remote Shell. Hier kommen als mögliche Authentifikationsmethoden $M1$ und $M2$/$M3$ (durch SSL/TLS mit Client-Authentifikation) in Frage.
- $L5$: Authentifikation über den SicAri-Web Server. Die Authentifikation auf Ebene $L5$ kann durch die HTTP-basierten Mechanismen *Basic-* oder *Digest-Authentication* [58] erfolgen. Bei beiden Verfahren erfolgt die Authentifikation mittels der Methode $M1$. Der Benutzername und das Password werden dazu in entsprechenden HTTP-Headern übertragen. Bei der Basic-Authentifikation werden diese im Klartext übertragen. Daher ist hier eine zusätzliche Verschlüsselung durch Verwendung des SSL/TLS-Protokolls sinnvoll. Die Authentifikation kann auch komplett über SSL/TLS abgewickelt werden, wenn hier die Option der Client-Authentifizierung ($M2/M3$) aktiviert wird.
- $L6$: Authentifikation über das SicAri-Web Service Framework. Hierbei sind die Authentifikationsmethoden $M1$-$M4$ möglich. Eine entsprechende Implementierung ist Gegenstand dieser Arbeit und wird in Kapitel 4.2 vorgestellt.

Die SicAri-Plattform sieht verschiedene Authentifikationsszenarien vor. Dabei wird unterschieden ob der Benutzer physisch vor der Plattform präsent ist oder sich über ein anderes Endgerät mit der Plattform verbindet. Das Endgerät ist dabei nicht in der Lage eine eigene Instanz der SicAri-Plattform auszuführen bzw. führt keine eigene Instanz der Plattform aus. Als Endgeräte kommen hier beispielsweise ein entfernter PC aber auch ein mobiles Endgerät in Frage. In Abhängigkeit einer lokalen oder globalen Authentifikation ergeben sich die folgenden Authentifikationsszenarien (vgl. [80]).

4.1.1.4 Authentifikationsszenarien

Im Folgenden werden die möglichen Authentifikationsszenarien vorgestellt. Zur Wahrung der Vertraulichkeit und Integrität der Kommunikation zwischen der PF und dem globalen AM wird das Protokoll SSL/TLS eingesetzt. Außerdem authentifizieren sich beide Instanzen beim Verbindungsaufbau wechselseitig mittels des SSL/TLS Protokolls. Dazu muss die aus SSL/TLS bekannte Option der Client-Authentifizierung aktiviert werden (vgl. auch Kapitel 4.1.1.6).

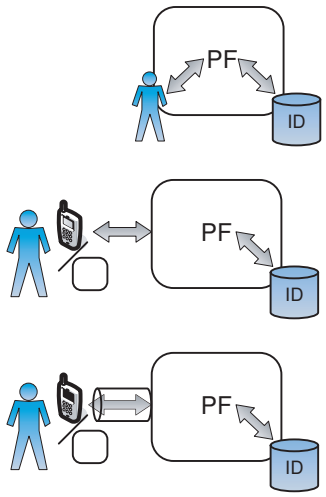

Abbildung 4.3: Lokale Authentifikationsszenarien $S1_{local} - S3_{local}$

Innerhalb von SicAri existieren drei lokale Authentifikationsszenarien, die in Abbildung 4.3 dargestellt sind.

- $S1_{local}$: Der Benutzer authentifiziert sich an der lokalen PF über die SicAri-Shell. Da keine Verbindung zum AM besteht, nutzt die PF lokal gespeicherte Informationen um die Identität und Authentizität des Benutzers festzustellen.

- $S2_{local}$: Der Benutzer authentifiziert sich über ein Endgerät bei der PF. Alle Verbindungen zwischen dem Endgerät und der PF sind nicht vor Verletzungen der Integrität oder Vertraulichkeit geschützt. Da keine Verbindung zum AM besteht, nutzt die PF lokal gespeicherte Informationen um die Identität und Authentizität des Benutzers festzustellen.

- $S3_{local}$: Der Benutzer authentifiziert sich über ein Endgerät bei der PF. Alle Verbindungen zwischen dem Endgerät und der PF sind vor Verletzungen der Vertraulichkeit und Integrität geschützt. Da keine Verbindung zum AM besteht, nutzt die PF lokal gespeicherte Informationen um die Identität des Benutzers festzustellen.

Die SicAri-PF erlaubt drei globale Authentifikationsszenarien, die in Abbildung 4.4 dargestellt sind.

- $S1_{global}$: Der Benutzer authentifiziert sich an der lokalen PF über SicAri-Shell. Die PF stellt dazu eine sichere Verbindung zum AM her. Dieser verifiziert die Identität und Authentizität des Benutzers anhand der, in der globalen Identitätsdatenbank gespeicherten Benutzerinformationen.

- $S2_{global}$: Der Benutzer authentifiziert sich über ein Endgerät bei der PF. Die PF stellt dazu eine sichere Verbindung zum AM her. Dieser verifiziert die Identität und Authentizität des Benutzers anhand der, in der globalen Identitätsdatenbank gespeicherten Benutzerinformationen. Die Verbindung zwischen der Plattform und dem Endgerät ist nicht vor Verletzungen der Integrität oder Vertraulichkeit geschützt.

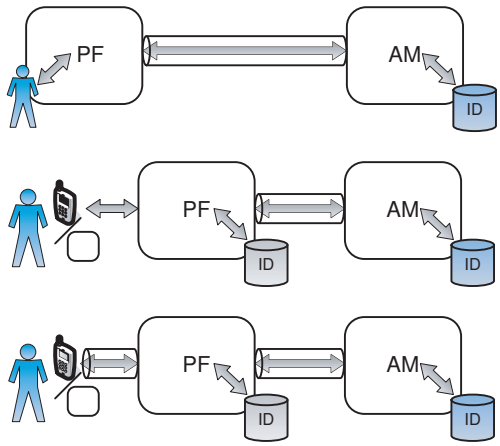

Abbildung 4.4: Globale Authentifikationsszenarien $S1_{global} - S3_{global}$

- $S3_{global}$: Der Benutzer authentifiziert sich über ein Endgerät bei der PF. Die PF stellt dazu eine sichere Verbindung zum AM her. Dieser verifiziert die Identität und Authentizität des Benutzers anhand der, in der globalen Identitätsdatenbank gespeicherten Benutzerinformationen. Die Verbindung zwischen der Plattform und dem Endgerät ist vor Verletzungen der Integrität oder Vertraulichkeit geschützt.

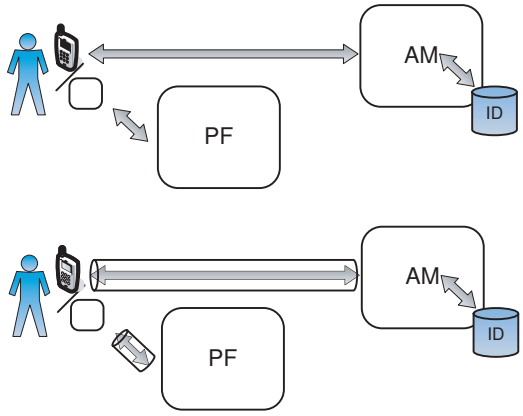

Abbildung 4.5: Entfernte Authentifikationsszenarien $S1_{remote} - S2_{remote}$

Außerdem existieren zwei entfernte (*remote*) Authentifikationsszenarien, die in Abbildung 4.5 dargestellt sind. Bei den entfernten Authentifikationsszenarien erfolgt die Authentifikation direkt beim AM. Im Gegensatz zu den lokalen oder globalen Authentifikationsszenarien wo das ST auf einer SicAri-Plattform sicher im Rahmen des `Subject`-Objekts verwahrt wird, wird das ST hier auf dem Endgerät selbst gespeichert. Danach kann der Benutzer das ST verwenden, um sich bei dem Aufruf von Services der SicAri-Plattform automatisch bei dieser zu authentifizieren. Hierbei muss das ST sicher von dem Endgerät zur Plattform übertragen werden. Zur Sicherung kommen hier die Anwendung des SSL/TLS-Protokolls in Frage. Außerdem können auf Ebene $L6$ auch Web Service-Security-Mechanismen eingesetzt werden. (vgl. [80])

- $S1_{remote}$: Der Benutzer authentifiziert sich zunächst über ein Endgerät beim AM. Dieser stellt daraufhin ein ST aus. Danach verbindet sich der Benutzer über sein Endgerät mit der SicAri-PF. Für die Authentifikation zeigt er sein ST vor. Alle Verbindungen zwischen dem Endgerät, der PF und dem AM sind nicht vor Verletzungen der Integrität oder Vertraulichkeit geschützt.

- $S2_{remote}$: Der Benutzer authentifiziert sich zunächst über ein Endgerät beim AM. Dieser stellt daraufhin ein ST aus. Danach verbindet sich der Benutzer über ein Endgerät mit der SicAri-PF. Für die Authentifikation zeigt er sein ST vor. Alle Verbindungen zwischen dem Endgerät, der PF und dem AM sind vor Verletzungen der Vertraulichkeit und Integrität geschützt.

4.1.1.5 Protokollspezifikation $AP1 : (S1_{global}, M1\text{-}M3, L1/L2)$

Im Rahmen dieser Arbeit wurde die globale Authentifikation auf Basis eines auf Web Services basierenden Protokolls umgesetzt. Dazu musste die Protokollspezifikation aus [80] teilweise angepasst werden. Auf die Unterschiede zwischen der SicAri-Protokollspezifikation und dem tatsächlich umgesetzten Protokoll wird anschließend eingegangen. Zunächst werden nun die einzelnen Protokollschritte des globalen Authentifikationsszenarien $S1_{global}$ vorgestellt. Dabei werden die Verfahren $M1$-$M3$ betrachtet. Zur Authentifikation kommen für dieses Szenario die Ebenen $L1$ und $L2$ in Frage. Die Tabelle 4.1 enthält zunächst die Syntax der Protokollspezifikation.

Es sei	U	der zu authentifizierende Benutzer
Es sei	id_U	die zur Identifikation eines Benutzers verwendete ID
Es sei	id_{PF}	die zur Identifikation einer Plattform verwendete ID
Es sei	$method_U$	das Verfahren, das zur Authentifikation eines Benutzers U verwendet wird
Es sei	$methods_U$	die möglichen Authentifikationsmethoden der Plattform PF für den Benutzer U
Es sei	$nonce_{AM}$	die vom Authentifikationsmanager AM erstellte Nonce
Es sei	$session_U$	ein Sitzungsidentifkator für den Benutzer U
Es sei	$token_{AM \rightarrow U}$	ein ST, das vom AM für den Benutzer U ausgestellt wurde
Es sei	M	eine Nachricht die zwischen zwei Instanzen ausgetauscht wird
Es sei	$hash_{AM}$	der vom Authentifikationsmanager AM definierte Typ des Hashalgorithmus
Es sei	$h(M)$	der Hash-Wert der Nachricht M
Es sei	$\{M, N\}$	eine Nachricht, die aus den Bestandteilen M und N zusammengesetzt ist
Es sei	K_U	der öffentliche Schlüssel des Benutzers U
Es sei	K_U^{-1}	der zugehörige private Schlüssel des Benutzers U
Es sei	$\{M\}_{K_U}$	die mit dem öffentlichen Schlüssel K_U verschlüsselte Nachricht M
Es sei	$\{h(M)\}_{K_U^{-1}}$	die mit dem privaten Schlüssel K_U^{-1} signierte Nachricht M
Es sei	$A \rightarrow B : \{M\}$	eine Nachricht M, die von A nach B gesendet wird
Es sei	$A \leftarrow B : \{M\}$	eine Nachricht M, die von B nach A gesendet wird

Tabelle 4.1: Syntax der Protokollspezifikation $AP1$

Die globale Authentifikation setzt sich aus folgenden Protokollschritten zusammen.

1. Der Benutzer U startet den Authentifikationsprozess.

2. Die Plattform PF fragt im Rahmen eines Login-Modul die Benutzer-ID id_U ab. In Abhängigkeit der Authentifikationsmethode werden vom Benutzer weitere Informationen abgefragt:

   (a) $M1$: Der Benutzer wird zur Eingabe seines Passworts aufgefordert.

   (b) $M2$: Der Benutzer wird nach seinem Key Store und dem zugehörigen Passwort, für den Zugriff auf den privaten Schlüssel K_U^{-1} aufgefordert.

   (c) $M3$: Der Benutzer wird zur Eingabe seiner PIN am Smart Card-Terminal, für den Zugriff auf den privaten Schlüssel K_U^{-1} aufgefordert.

3. Die Plattform PF stellt eine SSL/TLS-Verbindung zum Authentifikationsmanager AM her. Dabei wird die Authentizität von AM überprüft. Die Plattform PF sendet danach die Benutzer-ID an den AM, um die möglichen Authentifikationsmethoden abzufragen.

   (1) $PF \rightarrow AM : \{id_U\}$

4. AM überprüft die Authentizität der Plattform PF in dem es die Identitätsdatenbank auf das Vorhandensein der ID id_{PF} überprüft. Diese ID ergibt sich aus dem beim Verbindungsaufbau verwendeten SSL/TLS-Zertifikat. Dieses Zertifikat wird darüber hinaus auf Vertraulichkeit überprüft.

   Danach wird überprüft ob die Benutzer-ID id_U in der Datenbank existiert und welche Authentifikationsmethode die globale Plattform für diesen Benutzer anbietet. Existiert der Benutzer, generiert AM die Nonce $nonce_{AM}$. Diese wird zusammen mit den möglichen Authentifikationsmethoden $methods_U$ und dem für die weiteren Protokollschritte verwendeten Typ des Hash-Algorithmus $hash_{AM}$ an PF gesendet. Der Authentifikationsmanager AM speichert darüber hinaus die Werte $\{id_U, nonce_{AM}\}$ für die spätere Verifikation (vgl. Schritt 6) und zur Abwehr von Replay-Attacken.

   (2) $PF \leftarrow AM : \{methods_U, hash_{AM}, nonce_{AM}\}$

5. In Abhängigkeit der vom Authentifikationsmanager für den Benutzer U unterstützten Authentifikationsmethode $methods_U$ und der Auswahl des Benutzers, generiert die Plattform PF die Authentifikationsanfrage R:

   (a) $M1$: $R = \{h(h(password_U), nonce_{AM})\}$

   (b) $M2$: $R = \{h(id_U, nonce_{AM})\}_{K_U^{-1}}$

   (c) $M3$: $R = \{h(id_U, nonce_{AM})\}_{K_U^{-1}}$

   Bei der Berechnung von R für das Verfahren $M1$ wird der aus Protokollschritt 4 erhaltene Typ des Hash-Algorithmus dazu verwendet, den passenden Hash-Algorithmus für die Berechnung der Hash-Werte auszuwählen.

PF sendet diese mit dem ausgewählten Verfahren und der Benutzer-ID an den Authentifikationsmanager:

(3) $PF \rightarrow AM : \{method_U, id_U, R\}$

Hierbei wird wieder eine SSL/TLS-Verbindung zwischen PF und AM aufgebaut und die Authentizität der Beteiligten wechselseitig überprüft.

6. Der Authentifikationsmanager AM überprüft nun anhand der gewählten Authentifikationsmethode die Anfrage R. Der für diese Überprüfung benötigte öffentliche Schlüssel K_U (bei $M2$ & $M3$) bzw. der Hash-Wert des Passworts $h(password_U)$ (bei $M1$) bezieht der AM aus der Identitätsdatenbank. Die Nonce wird aus dem lokalen Speicher gelesen (vgl. Schritt 4).

   (a) $M1$: Es wird überprüft ob der Hash-Wert $h(h(password_U), nonce_{AM})$ mit der Anfrage R übereinstimmt

   (b) $M2$ & $M3$: Es wird überprüft ob $\{R\}_{K_U}$ mit dem Hash-Wert $h(id_U, nonce_{AM})$ übereinstimmt

   Konnten diese Werte erfolgreich überprüft werden, ist die Authentizität des Benutzers id_U nachgewiesen. Daraufhin wird ein ST ausgestellt und dieses mit dem privaten Schlüssel K_{AM}^{-1} des AM signiert:

   $token_{AM \rightarrow U} = \{token, K_{AM}\}_{K_{AM}^{-1}}$

   Dieses wird schließlich an die Plattform PF zurück gesendet.

   (4) $PF \leftarrow AM : \{token_{AM \rightarrow U}\}$

7. Die Plattform PF überprüft die Integrität, Vertrauenswürdigkeit und den Gültigkeitszeitraum des ST. Danach wird das ST als privates Credential im `Subject` gespeichert. Damit ist der Authentifikationsprozess abgeschlossen. Über das JAAS-Framework wird der Sicherheitskontext für den Benutzer erstellt und die Kontrolle an den Benutzer übergeben.

Vergleich zur SicAri-Protokollspezifikation Im Vergleich zur Protokollspezifikation aus [80] ergeben sich eine Reihe von Unterschieden, die hauptsächlich auf die Verwendung eines auf Web Services basierenden Protokolls zurückzuführen sind. Jede Verbindung zwischen PF und AM entspricht einem Web Service-Aufruf. Der AM kann somit nicht unterscheiden ob der zweite Aufruf im Protokoll zum ersten gehört. Dies kann nur durch die zusätzlich Angabe der Benutzer-ID entschieden werden. Daher ist das definierte Protokoll auch als zustandslos zu bezeichnen. Im Folgenden sind die einzelnen Unterschiede nochmals detailliert aufgelistet.

- Die Auswahl der Authentifikationsmethode erfolgt zunächst durch die Plattform bzw. durch den Benutzer. Ist an die lokale Plattform ein Smart Card-Lesegerät angeschlossen, startet das JAAS-Framework zunächst die Authentifikation mittels Smart Card. Besitzt der Benutzer keine Smart Card oder möchte diese nicht verwenden, wird nun die Authentifikation mittels Soft Token gestartet. Steht dem Benutzer kein Soft Token zur Verfügung

oder möchte er dieses nicht verwenden, wird schließlich die Authentifikation mittels Benutzername/Passwort gestartet. Hat sich der Benutzer für ein Verfahren entschieden, werden erst dann die möglichen Authentifikationsverfahren vom AM abgerufen. Anhand der möglichen Authentifikationsverfahren muss die lokale Plattform das vom Benutzer gewählte Verfahren ggf. abbrechen, wenn dieses vom AM nicht unterstützt wird. Dies ist auf die Implementierung des JAAS-Framework zurückzuführen. Im Kapitel 4.2.1 wird darauf noch detaillierter eingegangen.

- Bei jeder Kommunikation zwischen PF und AM muss eine SSL/TLS-Verbindung aufgebaut werden und die Authentizität der Partner wechselseitig überprüft werden.
- Bei der Abfrage der möglichen Authentifikationsverfahren sendet der AM neben dem Nonce auch noch den Typ des Hash-Algorithmus an die PF. Dieser wird bei der Authentifikation mittels Benutzername/Passwort zur Berechnung der Hash-Werte verwendet. Da für alle anderen Verfahren explizit keine Hash-Werte berechnet werden, wird dieser Parameter von diesen Verfahren entsprechend ignoriert.
- In Protokollschritt 5 wird zusätzlich die Benutzer-ID mit an den AM gesendet.

4.1.1.6 Plattform-Authentifikation

Unter der Plattform-Authentifikation wird die Authentifikation von SicAri-Plattformen untereinander verstanden. Kommunizieren SicAri-Plattformen miteinander, wird die Kommunikation normalerweise über das SSL/TLS-Protokoll abgesichert. Dieses Protokoll bietet die Möglichkeit der wechselseitigen Authentifikation mittels Zertifikaten. Standardmäßig wird das Server-Zertifikat an den Client übertragen. Dieser kann aufgrund des Zertifikats und der darin enthaltenen Informationen, wie beispielsweise den Aussteller des Zertifikats, entscheiden, ob er dem Server vertraut. Der Server kann zudem über eine Option des Protokolls auch ein Zertifikat vom Client anfordern. Anhand dessen kann der Server schließlich die Authentizität des Clients überprüfen.
Die Zertifikate erhalten die einzelnen SicAri-Plattformen von dem jeweiligen Plattform-Administrator, der die Plattform startet. Dazu muss sich der Plattform-Administrator mittels Soft Token oder Smart Card authentifizieren. Erfolgt die Authentifikation über Benutzername/Passwort oder ein Security Token, stehen keine Zertifikate für die Verwendung innerhalb von SicAri zur Verfügung. Dadurch kann die PF nur mit einer eingeschränkten Funktionalität betrieben werden.

4.1.2 Sicherheitsenforcement

Unter dem *Sicherheitsenforcement* wird die Durchsetzung der Sicherheitsrichtlinien innerhalb der SicAri-Plattform verstanden. Diese wird in Zusammenarbeit mit dem Policy-Framework (vgl. Kapitel 3.5.1) durchgesetzt. Das Policy-Framework wird in der SicAri eigenen Implementierung des Java-`SecurityManager` bei jeder Sicherheitsentscheidung konsultiert. Für welchen Benutzer diese Entscheidung getroffen werden muss, wird anhand des aktuellen Sicherheitskontext festgestellt. Dieser erlaubte es, zu überprüfen, welcher Benutzer gerade agiert. Wichtigster Vertrauensanker innerhalb der SicAri-Infrastruktur ist das für einen Benutzer ausgestellte ST. Wie bereits erwähnt ist ein ST innerhalb von SicAri ein Objekt das verschiedene Sicherheitsinformationen (vgl. Kapitel 3.5.3) in Form von Attributen enthält. Zusammen mit dem Policy-Framework kann in Abhängigkeit dieser Attribute beispielsweise entschieden werden, ob ein Zugriff auf sicherheitssensitive Informationen oder Dienste gewährt wird. Dazu prüft

die PF beispielsweise das Attribut, das die Login-Methode speichert. Für besonders sicherheitsrelevante Dienste kann der Zugriff beispielsweise nur dann erfolgen, wenn sich der Benutzer mittels des Smart Card-Verfahrens ($M3$) authentifiziert hat.
Das ST wird außerdem zur Realisierung des SSO innerhalb von SicAri verwendet. Ein Benutzer authentifiziert sich dazu zunächst mittels der Verfahren $M1$, $M2$ oder $M3$. Dadurch erhält er ein ST. Ist im Verlauf seiner Aktionen mit der PF eine erneute Authentifikation, beispielsweise durch den Aufruf eines Dienstes auf einer entfernten PF notwendig, so wird diese implizit mit dem ST ($M4$), realisiert. Eine PF kann anhand des ST entscheiden, ob sie dem sich damit authentifizierenden Benutzer vertraut oder nicht. Die PF entscheidet dies anhand der im ST gespeicherten Informationen. Die wichtigsten Informationen für die Entscheidung sind die ausstellende Instanz des ST, die gültige Signatur, die Authentifikationsmethode sowie der Gültigkeitszeitraum des ST. Vertraut die PF dem ST, so wird für den Benutzer auf der PF ein neuer Sicherheitskontext aufgebaut. Das geschieht durch eine Authentifikation des Benutzers auf Basis des ST ($M4$).

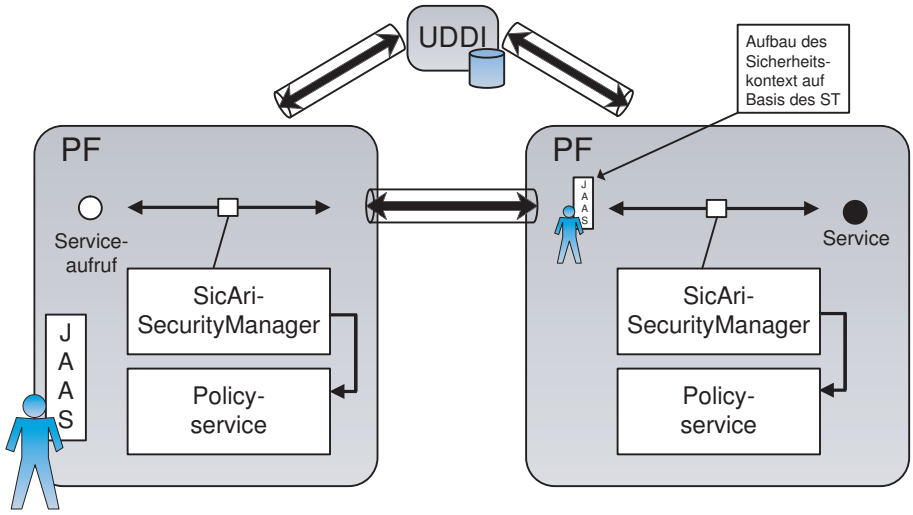

Abbildung 4.6: Beteiligte Komponenten des Sicherheitsenforcement

In Abbildung 4.6 ist dieser Zusammenhang noch einmal skizziert. Die Darstellung zeigt einen Benutzer, der sich an einer lokalen Plattform angemeldet hat und einen Dienst auf einer entfernten Plattform aufruft. Durch das ST wird der Benutzer auf der entfernten Plattform implizit authentifiziert und kann im Rahmen seines aufgebauten Sicherheitskontext den Dienst nutzen. Der detaillierte Ablauf dieses Vorgangs wird in der folgenden Protokollspezifikation vorgestellt.

4.1.2.1 Protokollspezifikation $AP2 : (S1_{global}, M4, L6)$

Die folgende Spezifikation definiert ein auf Web Services basierendes Protokoll zu Authentifikation eines Benutzers auf einer entfernten Plattform mittels eines ST. Für die weiteren Betrachtungen gilt neben dem in der Tabelle 4.1 festgelegten Syntax zudem der in Tabelle 4.2 dargestellte Syntax.

Es sei	PF_{local}	eine lokale SicAri-Plattform
Es sei	PF_{remote}	eine entfernte SicAri-Plattform

Es sei	$A \rightarrow B : \{M\}\{token_{AM \rightarrow U}\}$	eine Nachricht M, die von A nach B gesendet wird und ein vom AM für den Benutzer U ausgestelltes ST beinhaltet

Tabelle 4.2: Syntax der Protokollspezifikation $AP2$

Der Benutzer U hat sich an der lokalen PF angemeldet und wurde über den AM erfolgreich authentisiert. Als Ergebnis dieser Authentifikation wurde ein ST in dem zugehörigen `Subject` gespeichert. Der Benutzer möchte nun einen Service auf einer entfernten Plattform aufrufen. Dieser Service wurde in der UDDI-Registry registriert. Er steht damit für Zugriffe über die Web Service-Schnittstelle zur Verfügung:

1. Im Rahmen des Web Service-Zugriffs baut die lokale Plattform eine Verbindung zur entfernten Plattform auf. Diese ist durch SSL/TLS bzw. Web Service Security-Mechanismen vor Verletzungen der Vertraulichkeit und Integrität geschützt. Implizit wird das ST des Benutzers von der lokalen Plattform in die SOAP-Nachricht eingebettet und mit an die entfernte Plattform übertragen.

   (1) $PF_{local} \rightarrow PF_{remote} : \{M\}\{token_{AM \rightarrow U}\}$

2. Die entfernte Plattform nimmt die Anfrage entgegen und extrahiert das ST aus der SOAP-Nachricht. Danach verifiziert sie die Signatur des ST und überprüft ob sie der ausstellenden Instanz vertraut. Ist dies gegeben, wird noch der Gültigkeitszeitraum des ST überprüft. Auf Basis des ST wird auf der entfernten Plattform schließlich ein neuer Sicherheitskontext erstellt und der Zugriff auf den aufgerufene Dienst gewährt.

3. Der Service antwortet schließlich noch mit einer entsprechenden Response-Nachricht und sendet diese über PF_{remote} an das mobile Endgerät:

   (2) $PF_{local} \leftarrow PF_{remote} : \{N\}$

4.1.3 Mobile Endgeräte

Wie bereits erwähnt sieht die SicAri-Spezifikation den Zugriff von mobilen Endgeräten auf die Plattform vor. Hierfür ist es notwendig, dass der Benutzer, der sich über das mobile Endgerät mit der Plattform verbinden möchte, entsprechend authentifiziert wird.
Im Folgenden werden die mit einem mobilen Endgerät theoretisch möglichen Authentifikationsszenarien betrachtet. Hierbei wird davon ausgegangen, dass das mobile Endgerät nicht in der Lage ist, eine eigene SicAri-Plattform auszuführen. Der Zugriff auf die Plattform kann daher nur über definierte Schnittstellen wie die Remote-Shell oder den Web Server bzw. den Web Service Gateway erfolgen. Im Kapitel 4.2.9 wird schließlich untersucht, welche dieser Verfahren sich aufgrund der Beschränkungen des Endgeräts wirklich umsetzen lassen.
Als mögliche Authentifikationsmethoden kommen die aus Kapitel 4.1.1.3 bekannten Verfahren $M1$-$M4$ in Frage. Die möglichen Authentifikationsebenen erstrecken sich über den Login über die Remote-Shell ($L4$) sowie die Authentifikation über den Web Server ($L5$) oder das Web Service Framework ($L6$).

Szenario	Methoden	Ebene	Sicherheit
$S2_{local}$	$M1$-$M4$	$L4$-$L6$	/
$S3_{local}$	$M1$-$M4$	$L4/L5$	SSL/TLS
		$L6$	SSL/TLS, WSS
$S2_{global}$	$M1$-$M4$	$L4$-$L6$	/
$S3_{global}$	$M1$-$M4$	$L4/L5$	SSL/TLS
		$L6$	SSL/TLS, WSS
$S1_{remote}$	$M1$-$M4$	$L4$-$L6$	/
$S2_{remote}$	$M1$-$M4$	$L4/L5$	SSL/TLS
		$L6$	SSL/TLS, WSS

Tabelle 4.3: Mobile Authentifikation

Damit sind für den Zugriff von einem mobilen Endgerät auf die SicAri-Plattform theoretisch die in Tabelle 4.3 dargestellten Authentifikationszenarien möglich (vgl. Kapitel 4.1.1.4).
Die Sicherung der Vertraulichkeit und Integrität der ausgetauschten Daten zwischen dem mobilen Endgerät und der Plattform bzw. dem Authentifikationsmanager kann durch verschiedene Mechanismen realisiert werden. Einerseits kann mittels SSL/TLS eine Absicherung erfolgen. Zudem wäre auch die Absicherung mittels Web Service-Security (WSS)-Mechanismen denkbar. Theoretisch ist auch die Kombination zwischen SSL/TLS und WSS-Mechanismen möglich. Dies würde jedoch keinen Mehrwert bringen und die Leistungsfähigkeit unnötig verschlechtern. Es ist jedoch zu erwähnen, dass mit den WSS-Mechanismen differenziertere Sicherungen möglich sind. WSS erlaubt die Verschlüsselung und Signatur einzelner sicherheitsrelevanter Teile einer Nachricht und zudem eine Ende-zu-Ende-Sicherheit, SSL/TLS dagegen nur eine Punkt-zu-Punkt-Sicherheit (vgl. Kapitel 2.8.1). Jedoch ist SSL/TLS effizienter als WSS, da die Sicherung direkt auf der Transportebene erfolgt, anstatt durch die aufwendigere Verarbeitung und Sicherung von XML-Nachrichten.

4.1.3.1 Protokollspezifikation $AP3 : (S2_{remote}, M4, L6)$

Am Beispiel für einen Zugriff eines Benutzers auf einen entfernten SicAri-Service über die Web Service-Schnittstelle soll nun das Authentifikationsprotokoll dargestellt werden. Der Benutzer authentifiziert sich zunächst direkt beim AM und erhält ein ST.
Neben dem in der Tabelle 4.1 festgelegten Syntax gilt zudem der in Tabelle 4.4 dargestellte Syntax.

Es sei	PF_{mobile}	eine mobiles Endgerät
Es sei	$A \rightarrow B : \{M\}\{token_{AM \rightarrow U}\}$	eine Nachricht M, die von A nach B gesendet wird und ein vom AM für den Benutzer U ausgestelltes ST enthält

Tabelle 4.4: Syntax der Protokollspezifikation $AP3$

1. Der Benutzer U startet den Authentifikationsprozess.

2. Das mobile Endgerät PF_{mobile} stellt eine SSL/TLS-Verbindung zum Authentifikationsmanager AM her. Das mobile Endgerät PF_{mobile} sendet danach die Benutzer-ID an die Plattform, um die möglichen Authentifikationsmethoden abzufragen.

(1) $PF_{mobile} \rightarrow AM : \{id_U\}$

3. AM überprüft, ob die Benutzer-ID id_U in der Datenbank existiert und welche Authentifikationsmethode die globale Plattform für diesen Benutzer anbietet. Existiert der Benutzer, generiert AM die Nonce $nonce_{AM}$. Diese wird zusammen mit den möglichen Authentifikationsmethoden $methods_U$ und dem für die weiteren Protokollschritte verwendeten Hash-Algorithmus $hash_{AM}$ an PF_{mobile} gesendet. Der Authentifikationsmanager AM speichert darüber hinaus die Werte $\{id_U, nonce_{AM}\}$ für die spätere Verifikation (vgl. Schritt 5) und die Abwehr von Replay-Attacken.

   (2) $PF_{mobile} \leftarrow AM : \{methods_U, hash_{AM}, nonce_{AM}\}$

4. In Abhängigkeit der vom Authentifikationsmanager für den Benutzer U unterstützten Authentifikationsmethoden $methods_U$, der Auswahl des Benutzers und der technischen Möglichkeiten des mobilen Endgeräts, generiert das mobile Endgerät PF_{mobile} die Authentifikationsanfrage R:

   (a) $M1$: $R = \{h(h(password_U), nonce_{AM})\}$
   (b) $M2$: $R = \{h(id_U, nonce_{AM})\}_{K_U^{-1}}$
   (c) $M3$: $R = \{h(id_U, nonce_{AM})\}_{K_U^{-1}}$

   Bei der Berechnung von R für das Verfahren $M1$ wird der aus Protokollschritt 3 erhaltene Typ des Hash-Algorithmus dazu verwendet, den passenden Hash-Algorithmus für die Berechnung der Hash-Werte auszuwählen.

   PF_{mobile} sendet diese mit dem ausgewählten Verfahren und der Benutzer-ID an den Authentifikationsmanager:

   (3) $PF_{mobile} \rightarrow AM : \{method_U, id_U, R\}$

   Hierbei wird wieder eine SSL/TLS-Verbindung zwischen PF_{mobile} und AM aufgebaut.

5. Der Authentifikationsmanager AM überprüft nun anhand der gewählten Authentifikationsmethode die Anfrage R. Der für diese Überprüfung benötigte öffentliche Schlüssel K_U (bei $M2$ & $M3$) bzw. der Hash-Wert des Passworts $h(password_U)$ (bei $M1$) bezieht der AM aus der Identitätsdatenbank. Die Nonce wird aus dem lokalen Speicher gelesen (vgl. Schritt 2).

   (a) $M1$: Es wird überprüft, ob der Hash-Wert $h(h(password_U), nonce_{AM})$ mit der Anfrage R übereinstimmt
   (b) $M2$ & $M3$: Es wird überprüft, ob $\{R\}_{K_U}$ mit dem Hash-Wert $h(id_U, nonce_{AM})$ übereinstimmt

   Konnten diese Werte erfolgreich überprüft werden, ist die Authentizität des Benutzers id_U nachgewiesen. Daraufhin wird ein ST ausgestellt und dieses mit dem privaten Schlüssel K_{AM}^{-1} des AM signiert:

$token_{AM \rightarrow U} = \{token, K_{AM}\}_{K_{AM}^{-1}}$

Dieses wird schließlich an das mobile Endgerät PF_{mobile} zurückgesendet.

(4) $PF_{mobile} \leftarrow AM : \{token_{AM \rightarrow U}\}$

6. Damit ist der Authentifikationsprozess abgeschlossen. Das mobile Endgerät PF_{mobile} muss nun das erhaltene ST sicher verwahren und bei weiteren Zugriffen auf die Plattform vorzeigen.

Nach der erfolgreichen Authentifikation ruft der Benutzer über das mobile Endgerät einen Service der Plattform über die Web Service-Schnittstelle auf. Dabei zeigt er sein ST vor:

1. Im Rahmen des Web Service-Zugriffs baut das mobile Endgerät PF_{mobile} eine Verbindung zu einer Plattform auf. Diese ist durch SSL/TLS bzw. WS-Security-Mechanismen vor Verletzungen der Vertraulichkeit und Integrität abgesichert. Implizit wird das ST des Benutzers von dem mobile Endgerät PF_{mobile} in die SOAP-Nachricht eingebettet und mit an die Plattform übertragen.

   (1) $PF_{mobile} \rightarrow PF_{remote} : \{M\}\{token_{AM \rightarrow U}\}$

2. Die Plattform nimmt die Anfrage entgegen und extrahiert das ST aus der SOAP-Nachricht. Danach verifiziert sie die Signatur des ST und überprüft, ob sie der ausstellenden Instanz vertraut. Ist dies gegeben, wird noch der Gültigkeitszeitraum des ST überprüft. Auf Basis des ST wird auf der Plattform schließlich ein neuer Sicherheitskontext erstellt und der Zugriff auf den aufgerufene Dienst gewährt.

3. Der Service antwortet schließlich noch mit einer entsprechenden Response-Nachricht und sendet diese über PF_{remote} an das mobile Endgerät:

   (2) $PF_{mobile} \leftarrow PF_{remote} : \{N\}$

4.2 Implementierung

In diesem Kapitel wird nun detailliert auf die während dieser Arbeit vorgenommene Implementierung eingegangen. Zunächst erfolgt ein grober Überblick über die einzelnen Komponenten des Pakets `de.sicari.authentication` anhand des in Abbildung 4.7 dargestellten Klassendiagramms. Das Klassendiagramm zeigt zusätzlich wichtige Klassen aus anderen Paketen, die an dem in Klammern angegebene Paketnamen zu erkennen sind.
Eine der wichtigsten Komponente ist die Implementierung des Authentifikationsmanager, der für die Ausstellung eines globalen ST verantwortlich ist. Dieser Dienst arbeitet sehr eng mit den Authentifikationsdienst zusammen, der lokale STs ausstellt.
Eine entscheidende Komponente des globalen Authentifikationsprozess ist der Axis-Handler `AuthenticationManagerHandler`. Dieser verarbeitet SOAP-Nachrichten in Form eines `Document`-Objekts. Zu seinen Aufgaben gehört die Sicherung der Integrität, Vertraulichkeit

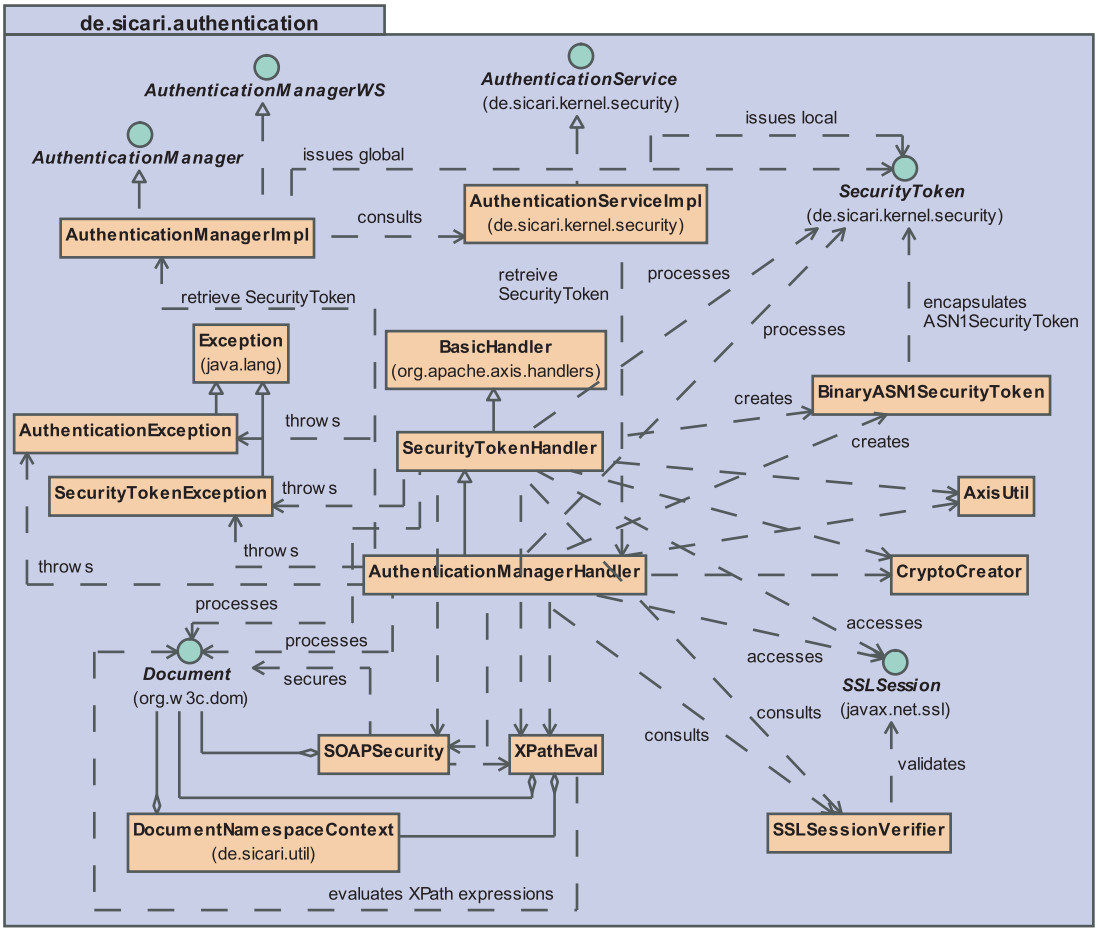

Abbildung 4.7: Klassendiagramm: Paket de.sicari.authentication

und ggf. auch der Authentizität der ausgetauschten Nachrichten zwischen einer lokalen Plattform und dem Authentifikationsmanager. Seine wichtigste Aufgabe ist jedoch die Übertragung des ausgestellten ST an die lokale Plattform.

Der `SecurityTokenHandler` ist ein weiterer Axis-Handler, der das ST bei einem Aufruf eines entfernten Service implizit, von der lokalen Plattform zur entfernten überträgt und so für den Aufbau eines neuen Sicherheitskontext auf dieser Plattform sorgt. Dieser Handler ist damit eine der Hauptkomponenten zur Realisierung des SSO innerhalb von SicAri.

Die Axis-Handler arbeiten mit einer Reihe von Hilfsklassen zusammen. So wird die Klasse `SSLSessionVerifier` zur Validierung einer SSL/TLS-Verbindung genutzt. Über die Klasse `SOAPSecurity` wird die Verschlüsselung und Signatur des SOAP-Body vorgenommen. Die Klasse `XPathEval` wird benötigt, um eine SOAP-Nachricht auf das Vorhandensein bestimmter XML-Elemente oder XML-Attribute zu analysieren und um auf diese zuzugreifen. Die Klasse `BinaryASN1SecurityToken` dient dazu, ein `ASN1SecurityToken` bzw. `ASN1EncryptedSecurityToken` in eine SOAP-Nachricht einzubetten.

Im Folgenden werden nun die einzelnen Komponenten detailliert vorgestellt.

4.2.1 Authentifikation

Bei der Authentifikation von Benutzern innerhalb von SicAri sind neben dem JAAS-Framework eine Reihe von weiteren Klassen beteiligt. Das Zusammenspiel dieser ist in dem Klassendiagramm 4.8 dargestellt. Wie bereits erwähnt, steuert die Klasse `LoginContext` den generellen Ablauf der Authentifikation, indem sie die einzelnen `LoginModule`-Klassen initialisiert und das `Subject`-Objekt erstellt. Die Klasse `LoginContext` wird selbst innerhalb der Methode `authenticate(...)` der Klasse `AuthenticationServiceImpl` initialisiert. Innerhalb dieser Klasse werden auch die erwähnten Objekte `CallbackHandler` und `Configuration` erstellt.
Die `AuthenticationServiceImpl`-Klasse bildet den Dienst für die lokale Benutzerauthentifikation. Die Hauptaufgabe dieser Klasse ist das Starten der Authentifikation sowie das Ausstellen von STs. Die Klasse implementiert das `AuthenticationService`-Interface, das die Schnittstelle für den Zugriff auf diesen Dienst bildet.

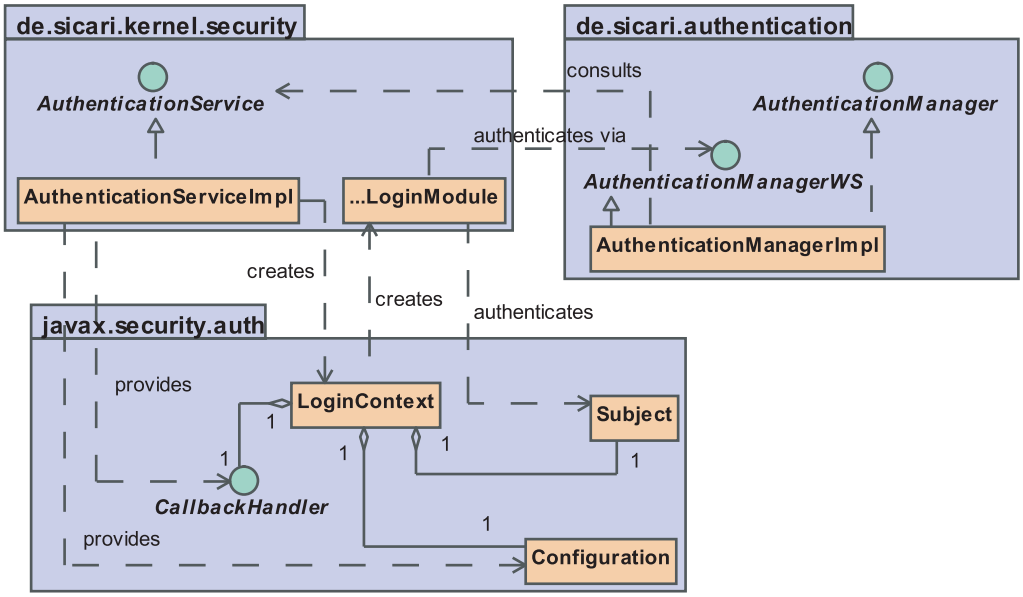

Abbildung 4.8: Klassendiagramm: Zusammenspiel von JAAS und SicAri

Konfiguration Im Rahmen dieser Arbeit wurde für die Konfiguration des Authentifikationsdienst die Datei `authentication.conf` angelegt. Dieser Dienst kennt dabei die folgenden Konfigurationsparameter.

- `login.profile`: Über diesen Parameter wird das zu verwendete Login-Profil für das JAAS-Framework ausgewählt. Standardmäßig wird hier der Wert `default` angegeben und damit das Profil `default` ausgewählt, das in der Konfigurationsdatei `login.conf` (siehe Kapitel E.2.1) spezifiziert ist.
- `token.type`: Dieser Parameter legt den Typ des ST fest. Damit wird spezifiziert welches ST vom `AuthenticationService`-Dienst ausgestellt werden soll. Standardmäßig wird ein `ASN1SecurityToken` verwendet. Dies wird durch Angabe des Werts

`asn1` festgelegt. Soll dieser Dienst stattdessen ein `SAMLSecurityToken` bzw. ein `XMLSecurityToken` ausstellen, so muss dies durch den Wert `saml` bzw. `xml` angegeben werden.

- `token.validity`: Dieser Parameter legt die Gültigskeitslaufzeit eines ausgestellten ST fest. Die Angabe erfolgt in Minuten. Als Standardwert wird hier eine Gültigkeit von 60 Minuten festgelegt.
- `callback.handler`: Über diese Parameter wird der Typ des `CallbackHandler`, der für das Abfragen von beispielsweise Benutzernamen und Passwörtern innerhalb des JAAS-Frameworks verwendet werden soll. Der Standardwert für diesen Parameter ist `com.sun.security.auth.callback.TextCallbackHandler`. Dadurch erfolgt die Auswahl eines `CallbackHandler` mit Text-basierter Eingabe auf der Shell.

Ein Beispiel für diese Konfigurationsdatei ist im Anhang in Kapitel E.1.1 aufgeführt.

Login-Module: Im gleichen Paket wie der Authentifikationsdienst sind außerdem die einzelnen Login-Module, die die Plattform anbietet, implementiert. Das Klassendiagramm 4.9 stellt den Aufbau der Login-Module dar. Die Basis für alle SicAri-Login-Module bildet die abstrakte Klasse *AbstractLoginModule*, die das von dem JAAS-Framework definierte Interface *LoginModule* implementiert. Die wichtigsten Methoden dieser Klasse und deren Funktionen und Abfolge sind:

- `initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState,Map options)`: Mit dieser Methode wird das jeweilige Login-Modul mit den in der Klasse `LoginContext` erstellten Parametern initialisiert.
- `boolean login()`: Mit dieser Methode wird die Authentifikation des Benutzers gestartet. Sie ruft intern die Methode `authenticate()` auf.
- `UID authenticate()`: Diese Methode ist als abstrakt deklariert und wird erst in den konkreten Login-Modulen implementiert. Innerhalb dieser Methode wird die Authentifikation des Benutzers implementiert.
- `UID requestLoginName()`: Mit dieser Methode wird der Login-Name des Benutzers abgefragt.
- `boolean commit()`: Diese Methode dient dazu, die im Verlauf des Authentifikationsprozesses erstellten `Principal`-Objekte sowie privaten und öffentlichen Credentials dem `Subject` hinzuzufügen.
- `boolean logout()`: Mit dieser Methode beendet ein Benutzer seine bestehende Sitzung und erreicht damit die Zerstörung aller sicherheitsrelevanten Objekte (`Principal` und Credentials) des `Subject`-Objekts.
- `boolean abort()`: Mit dieser Methode wird der Anmeldeprozess eines Benutzers im Fehlerfall abgebrochen. Alle bis dahin bereits erstellten `Principal`-Objekte und Credentials werden dabei gelöscht.

Nach dem die Authentifikation anhand der im Folgenden beschriebenen Login-Module abgeschlossen ist, überprüft der Authentifikationsdienst, ob das `Subject`-Objekt schon ein vom Authentifikationsmanager ausgestelltes ST enthält. Ist dies nicht der Fall, stellt er ein lokales ST aus und fügt es dem `Subject`-Objekt hinzu. Danach startet er für den Benutzer eine Shell.

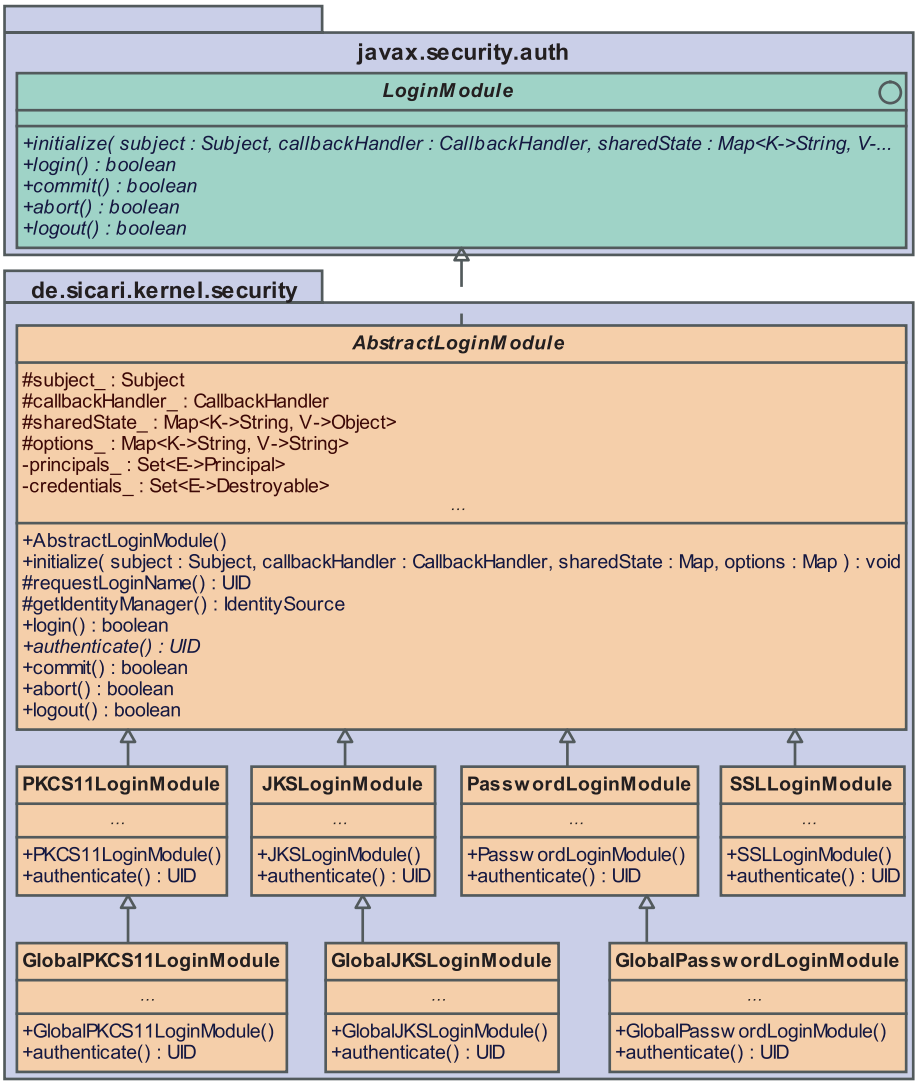

Abbildung 4.9: Klassendiagramm: SicAri-Login-Module

4.2.1.1 Lokale Authentifikation

Die konkreten Implementierungen der Login-Module für die lokale Authentifikation umfassen die Klassen `PKCS11LoginModule`, die die Authentifikation mittels Smart Card realisiert. Außerdem die Klasse `JKSLoginModule` mit der die Authentifikation mittels Soft Token umgesetzt wird. Mit der Klasse `PasswordLoginModule` wird eine Benutzername/Passwort-basierende Authentifikation ermöglicht. Für den Login über die SicAri-Remote-Shell existiert noch die Klasse `SSLLoginModule`, mit der eine Authentifikation durch das bei der SSL/TLS-Verbindung angeforderte Client-Zertifikat realisiert wird.

Die Abläufe der einzelnen `authenticate()`-Methoden der jeweiligen Login-Module ähneln sich. Zunächst wird der Benutzername durch Aufruf der Methode `requestLoginName()` ermittelt. Danach wird bei der Smart Card- bzw. Soft Token-basierenden Authentifikation überprüft, ob zu dem angegebenen Namen des Benutzers Zertifikate in der lokalen Identitätsdaten-

bank existieren. Ist dies der Fall, so wird bei Anwendung des *PKCS11LoginModule* der private Schlüsselmanager erstellt. Dieser interagiert direkt über einen so genannten PKSC11-Provider der JCA-Architektur mit dem angeschlossenen Smart Card-Lesegerät und ermöglicht so den Zugriff auf die kryptografischen Funktionen der Smart Card. Bei Nutzung des `JKSLoginModule` wird das Soft Token des Benutzers und das zugehörige Passwort abgefragt und danach schließlich der private Schlüsselmanager erstellt.
Bei beiden Verfahren wird beim Erstellen des Schlüsselmanagers anhand der darin enthaltenen öffentlichen Zertifikate überprüft, ob diese mit den Zertifikaten übereinstimmen, die unter dem angegebenen Benutzernamen in der Identitätsdatenbank zu finden sind. Zusätzlich wird überprüft, ob diese vertrauenswürdig sind. Nach der Abfrage der PIN (der Smart Card) bzw. des Passworts (des Soft Token) wird jeweils überprüft, ob der jetzt zugängliche private Schlüssel mit dem öffentlichen Schlüssel in der Identitätsdatenbank korrespondiert. Dazu wird mit dem privaten Schlüssel eine Signatur erstellt und diese mit dem in der Datenbank gespeicherten öffentlichen Schlüssel überprüft.
Im Fall des `PasswordLoginModule` wird das Passwort des Benutzers abgefragt und überprüft, ob der berechnete Hash-Wert des Passworts mit dem in der Datenbank gespeicherten Wert korrespondiert.
Liefern diese Überprüfungen jeweils positive Ergebnisse, so ist der Benutzer erfolgreich authentifiziert. Letztlich werden noch der private Schlüsselmanager (nur bei Smart Card bzw. Soft Token) und die Authentifikationsmethode gespeichert. Diese werden dann später mittels der `commit()`-Methode im `Subject` gespeichert.

4.2.1.2 Globale Authentifikation

Für die globale Authentifikation wurden die bestehenden Login-Module um drei weitere ergänzt. Die Klassen `GlobalPasswordLoginModule`, `GlobalPKCS11LoginModule` sowie `GlobalJKSLoginModule` erben dabei die Eigenschaften und Methoden der korrespondierenden lokalen Login-Module (vgl. Abbildung 4.10).
Der Ablauf der globalen Authentifikation bei Verwendung eines Soft Token ist in Abbildung 4.10 als Sequenzdiagramm dargestellt. Im Folgenden wird dieser Ablauf beschrieben. Die Sequenzdiagramme für die Smart Card- bzw. Benutzername/Passwort-basierende Authentifikation besitzen einen ähnlichen Aufbau und werden hier nicht weiter detailliert.
Zunächst wird innerhalb der `authenticate()`-Methode über das Environment überprüft, ob der globale Authentifikationsmanager verfügbar ist. Dieses wird über die Klasse `Lookup` realisiert. Über die Methode `environmentWhatIs(...)` wird der AM anhand dem durch `WhatIs` (vgl. Kapitel 4.3.1.1) festgelegten Pfad gesucht. Da der AM im Environment der lokalen PF nicht verfügbar ist, wird implizit eine Suche in der SicAri-UDDI-Registry vorgenommen. Ist der Authentifikationsmanager verfügbar, so wird automatisch der WS-Client-Stub generiert und dieser schließlich auf der lokalen Plattform im Environment publiziert. Dadurch wird der Authentifikationsmanager über den lokalen WS-Client-Stub bei der nächsten Suche direkt im lokalen Environment gefunden und es muss somit nicht erneut die UDDI-Registry konsultiert und der WS-Client-Stub generiert werden. Danach wird, wie auch bei der lokalen Authentifikation, der Benutzername abgefragt. Hierbei wird jedoch nicht überprüft, ob dieser der lokalen Plattform bekannt ist.
Bei der Smart Card-basierenden Authentifikation wird nun der private Schlüsselmanager erstellt. Bei der Soft Token-basierenden Authentifikation wird zunächst das Soft Token des Benutzers sowie das zugehörige Passwort abgefragt und danach erst der private Schlüsselmanager erstellt. Bei der Erstellung des privaten Schlüsselmanagers wird hier im Gegensatz zur lokalen

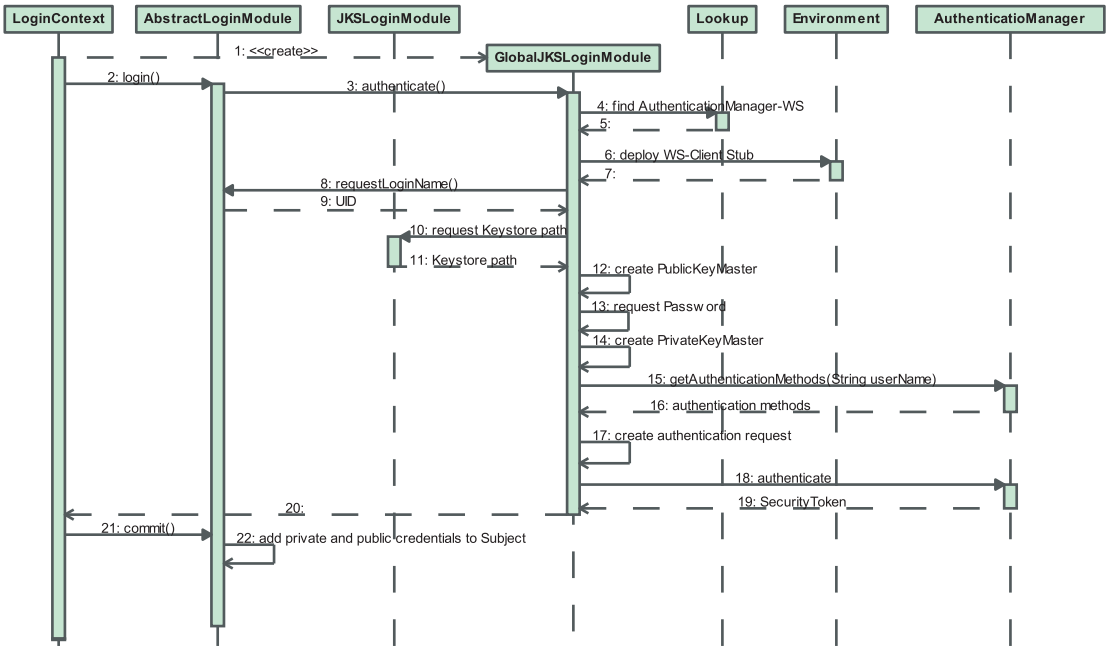

Abbildung 4.10: Sequenzdiagramm: Globale Authentifikation mittels Soft Token

Authentifikation jedoch nicht überprüft, ob die zugehörigen öffentlichen Zertifikate mit denen in der lokalen Datenbank übereinstimmen. Außerdem wird auch nicht überprüft, ob die privaten Schlüssel mit denen in der lokalen Datenbank korrespondieren. Diese Überprüfungen erfolgen nun über den globalen Authentifikationsmanager.
Zunächst werden über die Methode `getAuthenticationMethods(...)` des Authentifikationsmanagers die möglichen Authentifikationsverfahren abgefragt. Neben den möglichen Verfahren beinhaltet die Antwort des Authentifikationsmanagers auch eine Nonce sowie den Typ des Hash-Algorithmus, der bei der Berechnung des Passwort-Hash-Werts verwendet werden muss. Anhand der möglichen Authentifikationsverfahren wird nun ggf. die Authentifikation innerhalb des aktuellen Login-Module abgebrochen, falls diese nicht unterstützt wird.
Bei der Smart Card- bzw. Soft Token-basierenden Authentifikatikon wird nun die Authentifikationsanfrage berechnet. Dazu signiert die PF den Benutzernamen und die Nonce mit dem privaten Schlüssel des Benutzers.
Bei Verwendung des `GlobalPasswordLoginModule` wird die Authentifikationsanfrage anhand des Benutzerpassworts berechnet. Dazu wird zunächst das Passwort vom Benutzer abgerufen und danach mit dem vorher erhaltenen Typ des Hash-Algorithmus der Hash-Wert davon berechnet. Dann wird von diesem Hash-Wert und der erhaltenen Nonce erneut ein Hash-Wert berechnet.
Nun wird die Methode `authenticate(...)` des Authentifikationsmanagers aufgerufen. Dabei wird das gewählte Authentifikationsverfahren, der Benutzername und die berechnete Authentifikationsanfrage angegeben. Konnte der Benutzer erfolgreich authentifiziert werden, antwortet der Authentifikationsmanager mit dem ausgestellten ST, das im letzten Schritt im `Subject`-Objekt gespeichert wird.
Vor der genauen Betrachtung des Authentifikationsmanagers erfolgt zunächst noch eine detaillierte Vorstellung der einzelnen Implementierungen der Security Tokens.

4.2.1.3 SecurityToken

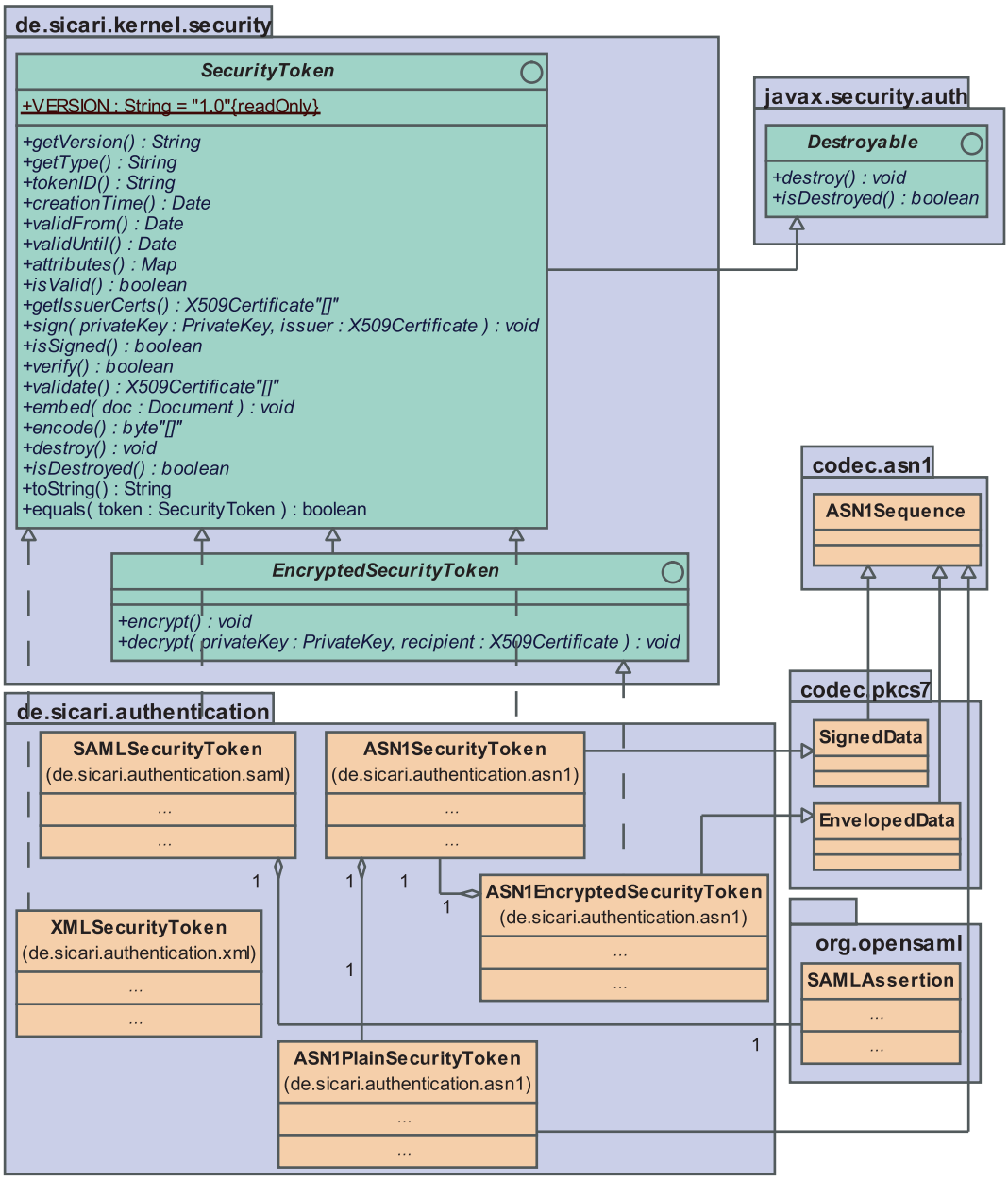

Abbildung 4.11: Klassendiagramm: SecurityToken-Übersicht

Das `SecurityToken`-*Interface* legt die Basismethoden eines ST fest. Diese Schnittstelle erbt selbst von dem Interface `Destroyable` des JAAS-Frameworks zwei Methoden. Dieses Interface dient dazu, das als privates Credential gespeicherte ST beim Abmelden eines Benutzers zu eliminieren. Das Interface `EncryptedSecurityToken` erweitert das `SecurityToken`-Interface um Methoden, die das Chiffrieren und Dechiffrieren eines ST ermöglichen.
Das `SecurityToken`-Interface besitzt die folgenden Methoden.

- `Map attributes()`: liefert eine Menge von Attributen, die für das ST definiert wurden.
- `Date creationTime()`: liefert das Datum an dem das ST ausgestellt wurde.
- `void destroy()`: Diese von dem Interface `Destroyable` des JAAS-Framework geerbte Methode, löscht alle sicherheitsrelevanten Informationen des ST.
- `void embed(Document doc)`: Mit dieser Methode wird das ST in das angegebene `Document` eingebettet.
- `byte[] encode()`: Diese Methode liefert das ST in einer `Base64`-kodierten Darstellung zurück.
- `boolean equals(SecurityToken token)`: Diese Methode vergleicht zwei ST auf Übereinstimmung.
- `X509Certificate[] getIssuerCerts()`: Diese Methode liefert die X509-Zertifikate, die erfolgreich validiert werden konnten und zur Signierung des ST verwendet wurden.
- `String getType()`: liefert den implementierten Typ (`asn1`, `saml` oder `xml`) des ST.
- `String getVersion()`: liefert die aktuelle Versionsnummer des vorliegenden ST.
- `boolean isDestroyed()`: Mit dieser von dem Interface `Destroyable` des JAAS-Framework geerbten Methode kann überprüft werden, ob alle sicherheitsrelevanten Methoden des ST bereits gelöscht wurden.
- `boolean isSigned()`: Mit dieser Methode kann überprüft werden, ob das ST bereits signiert wurde.
- `boolean isValid()`: Mit dieser Methode kann überprüft werden, ob das ST schon gültig ist und nicht abgelaufen ist.
- `void sign(PrivateKey privateKey, X509Certificate issuer)`: Mit dieser Methode kann ein vorliegendes ST unter Verwendung des angegeben privaten Schlüssels signiert werden. Das zugehörige öffentliche Zertifikat des Aussteller wird in die Signatur integriert.
- `String tokenID()`: liefert die ID des ST.
- `String toString()`: liefert eine Repräsentation des ST als Zeichenkette.
- `X509Certificate[] validate()`: Diese Methode erlaubt es, die Zertifikate des Ausstellers der Signatur auf Vertrauenswürdigkeit zu überprüfen. Dabei wird die lokale Schlüsselverwaltung konsultiert und überprüft, ob der Aussteller des Zertifikats der Plattform bekannt ist und dieser als vertrauenswürdig eingestuft wird.
- `Date validFrom()`: liefert den Zeitpunkt, ab dem das ST gültig ist.
- `Date validUntil()`: liefert den Zeitpunkt, bis zu dem das ST gültig ist.

- `boolean verify()`: Mittels dieser Methode wird die Signatur eines ST verifiziert. Dazu werden die öffentlichen Zertifikate verwendet und bei erfolgreicher Verifikation in einer internen Liste gespeichert. Diese können mit der Methode `validate()` auf ihre Vertrauenswürdigkeit überprüft werden.

Das `EncryptedSecurityToken`-*Interface* erbt alle Methoden von dem `SecurityToken`-Interface und erweitert es um zwei Methoden.

- `void decrypt( PrivateKey key, X509Certificate recipient)`: Mit dieser Methode wird ein verschlüsseltes ST unter Verwendung des angegebenen privaten Schlüssels und des zugehörigen Zertifikats entschlüsselt.
- `void encrypt()`: Diese Methode verschlüsselt das `SecurityToken`. Eine implementierende Klasse muss es ermöglichen, die öffentlichen Schlüssel des Empfängers für die Verschlüsselung des ST entgegenzunehmen.

Derzeit existieren in der SicAri-Infrastruktur drei unterschiedliche Typen von Security Tokens (vgl. Klassendiagramm 4.11). Alle drei Typen implementieren das Interface `SecurityToken`. Das `SAMLSecurityToken` realisiert auf Basis einer SAML-Assertion die ST-Funktionen. Das `XMLSecurityToken` implementiert die Funktionalitäten auf Basis eines plattformeigenen proprietären XML-Formats. Die Verschlüsselung und Signatur dieser STs wird mittels den bekannten Mechanismen der XMLdsig- bzw. XMLenc-Spezifikation realisiert.
Mit dem `ASN1SecurityToken` wird eine ASN.1-kodierte Form des ST realisiert. Dieses ST besteht selbst aus einem `ASN1PlainSecurityToken`, das eine Sequenz von ASN1-kodierten Daten darstellt. Die ASN.1-Funktionen erbt das `ASN1PlainSecurityToken` von der Klasse `ASNSequence`. Das `ASN1SecurityToken` erbt von der Klasse `SignedData`-Funktionen um eine Signatur des ST zu ermöglichen. Für die Verschlüsselung eines ASN.1-basierenden ST existiert die Klasse `ASN1EncryptedSecurityToken`. Diese implementiert das `EncryptedSecurityToken`-Interface und erbt zudem die Funktionen für die Verschlüsselung von der Klasse `EnvelopedData`.
Die Abbildung 4.12 stellt die weiteren Methoden dieser ST-Implementierungen als Klassendiagramm dar. Die Methoden, die die einzelnen STs von ihrem Interface geerbt haben, sind zur besseren Übersicht hier nicht erneut aufgeführt.
Jedes ST besitzt zu seiner Erstellung entsprechende Konstruktoren. Diese umfassen jeweils einen Konstruktoren mit dem die ID eines ST, der Gültigkeitszeitraum sowie eine Reihe von Attributen in Form einer `Map` festgelegt werden können. Zudem besitzt jedes ST einen Kopier-Konstruktor. Damit kann ein vorhandenes ST eines anderen Typs in diesen Typ gewandelt werden. Das neue ST wird dabei mit dem Werten des alten ST initialisiert. Hierbei wird auch der Zeitpunkt der Erstellung des ST übernommen.
Beim Anlegen eines `ASN1SecurityToken` werden die angegebenen Werte verwendet, um ein `ASN1PlainSecurityToken` zu erzeugen. Das `ASN1EncryptedSecurityToken` besitzt außerdem Konstruktoren, um die Algorithmenparameter und den/die Empfänger der Verschlüsselung festzulegen. Das `SAMLSecurityToken` und das `XMLSecurityToken` besitzen zusätzlich Konstruktoren, um dieses ST anhand einer SAML-Assertion bzw. einem XML-Element zu initialisieren.
Alle STs besitzen eine Reihe von weiteren internen Methoden, auf die hier nicht näher eingegangen wird. Für weitere Informationen sei auf die entsprechenden *JavaDoc*-Kommentare verwiesen. Zwei Methoden des `ASN1SecurityToken` sollen jedoch noch erwähnt werden:

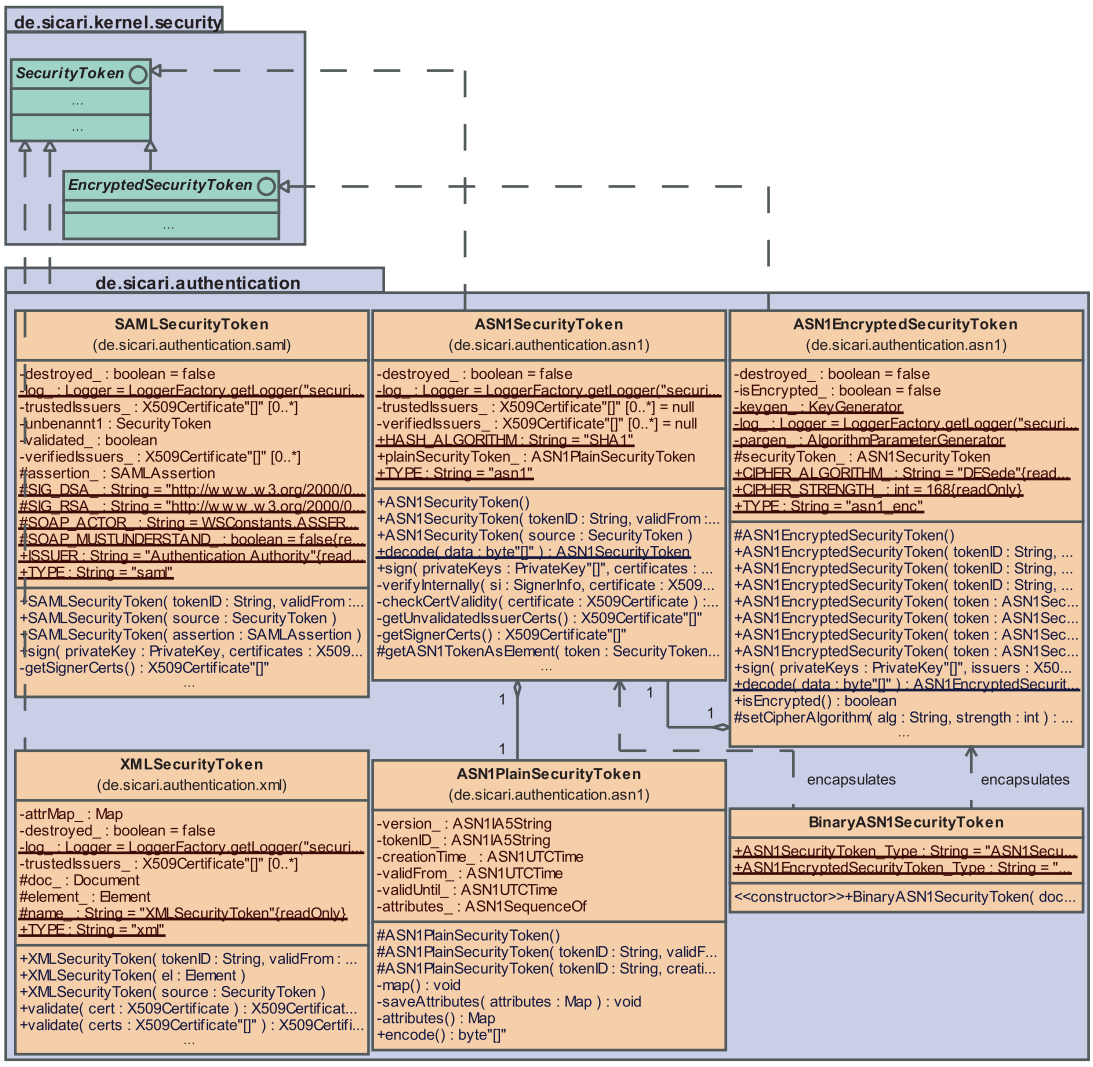

Abbildung 4.12: Klassendiagramm: SecurityToken-detailliert

- Die `getANS1TokenAsElement(...)`-Methode dient dazu, ein ASN.1-ST in ein XML-Element zu wandeln. Dieses XML-Element repräsentiert das in der WSS-Spezifikation definierte `<wsse:BinarySecurityToken>`-Element (siehe Kapitel 2.8.3.1). Für die Erzeugung dieses Elements wird die Klasse `BinaryASN1SecurityToken` verwendet. Diese Klasse spezialisiert die Klasse `BinarySecurity`, die aus der *WSS4J*-Implementierung[1] stammt. Das ST wird zunächst mittels der `encode()`-Methode in eine binäre Base64-Darstellung überführt und mit der Methode `setToken` als Kindelement eingefügt. Danach wird mit der Methode `setEncodingType` festgelegt, dass das ST in Base64-Kodierung gespeichert ist. Mit der Methode `setValueType` wird schließlich noch spezifiziert, ob es sich um ein `ASN1SecurityToken` oder um ein `ASN1EncryptedSecurityToken` handelt.

[1] `http://ws.apache.org/wss4j`

- Die `decode(byte[])`-Methode dient dazu, ein in binärer Darstellung vorliegendes ST wieder in ein `ASN1EncryptedSecurityToken`- bzw. `ASN1SecurityToken`-Objekt zu konvertieren.

4.2.1.4 Authentifikationsmanager

Der Authentifikationsmanager ist, wie bereits erwähnt, für die globale Authentifikation von Benutzern zuständig. Dieser Dienst ist kein alleinstehender Server, sondern ein Service, der auf einer speziellen SicAri-Plattform installiert ist. Der AM wird beim Starten einer Plattform als Web Service veröffentlicht und steht damit für den Zugriff über die WS-Schnittstelle zur Verfügung. SicAri bietet für die automatische Bereitstellung von Diensten als Web Service entsprechende Funktionen an. In Kapitel 4.3.2 wird vorgestellt wie diese verwendet werden.
Der AM wird durch die Klasse `AuthenticationManagerImpl` implementiert. Das Klassendiagramm in Abbildung 4.13 zeigt den Aufbau dieser Klasse sowie den Aufbau der Klasse `AuthenticationService` mit der dieser Dienst zusammenarbeitet. Zur besseren Übersicht sind die Methoden, die diese Klassen von ihren Schnittstellen erben, nicht erneut aufgeführt.
Die Klasse `AuthenticationManagerImpl` implementiert zwei Schnittstellen:

- `AuthenticationManager`: Dieses Interface definiert eine spezielle Zugriffsschnittstelle. Die spezifizierten Methoden werden ausschließlich von dem im nachfolgenden Kapitel vorgestellten Axis-Handler `AuthenticationManagerHandler` verwendet, um STs im Authentifikationsmanager zu speichern oder von diesem abzurufen (vgl. Kapitel 4.2.3).
- `AuthenticationManagerWS`: Mit diesem Interface werden die Methoden festgelegt, die später als WS zur Verfügung gestellt werden:
  - `byte[] getAuthenticationMethods(String userID)`:
    Mit dieser Methode wir der Authentifikationsprozess für den angegebenen Benutzer gestartet. Der Rückgabeparameter ist ein Byte-Array, das an Position 0 die möglichen Authentifzierungsverfahren enthält. Position 1 definiert den Typ des Hash-Algorithmus und der Rest des Array enthält die vom Authentifikationsmanager berechnete Nonce. Über diese Methode werden die Schritte 3 und 4 der Protokollspezifikation $AP1$ realisiert.
  - `String authenticate(byte m, String userID, byte[] req)`:
    Mit dieser Methode wird der Benutzer mit der angegebenen Benutzer-ID authentisiert. Neben dem gewählten Authentifikationsverfahren muss außerdem die Authentifikationsanfrage angegeben werden. Der Rückgabeparameter ist eine ID, mit der das ausgestellte ST identifiziert werden kann. Über diese Methode werden die Schritte 5 und 6 der Protokollspezifikation $AP1$ realisiert.
  - `void cancel(String userID)`:
    In einem Fehlerfall kann der Authentifikationsprozess mit dieser Methode abgebrochen werden.
  - `String authenticate(String tokenId)`:
    Der Authentifikationsmanager bietet mit dieser Methode die Möglichkeit an, sich mittels einem von einer lokalen Plattform ausgestellten ST zu authentifizieren. Der angegeben Parameter identifiziert das ST, das zur Authentifikation verwendet werden soll. Auch hier ist der Rückgabeparameter eine ID, mit der das ausgestellte ST identifiziert werden kann.

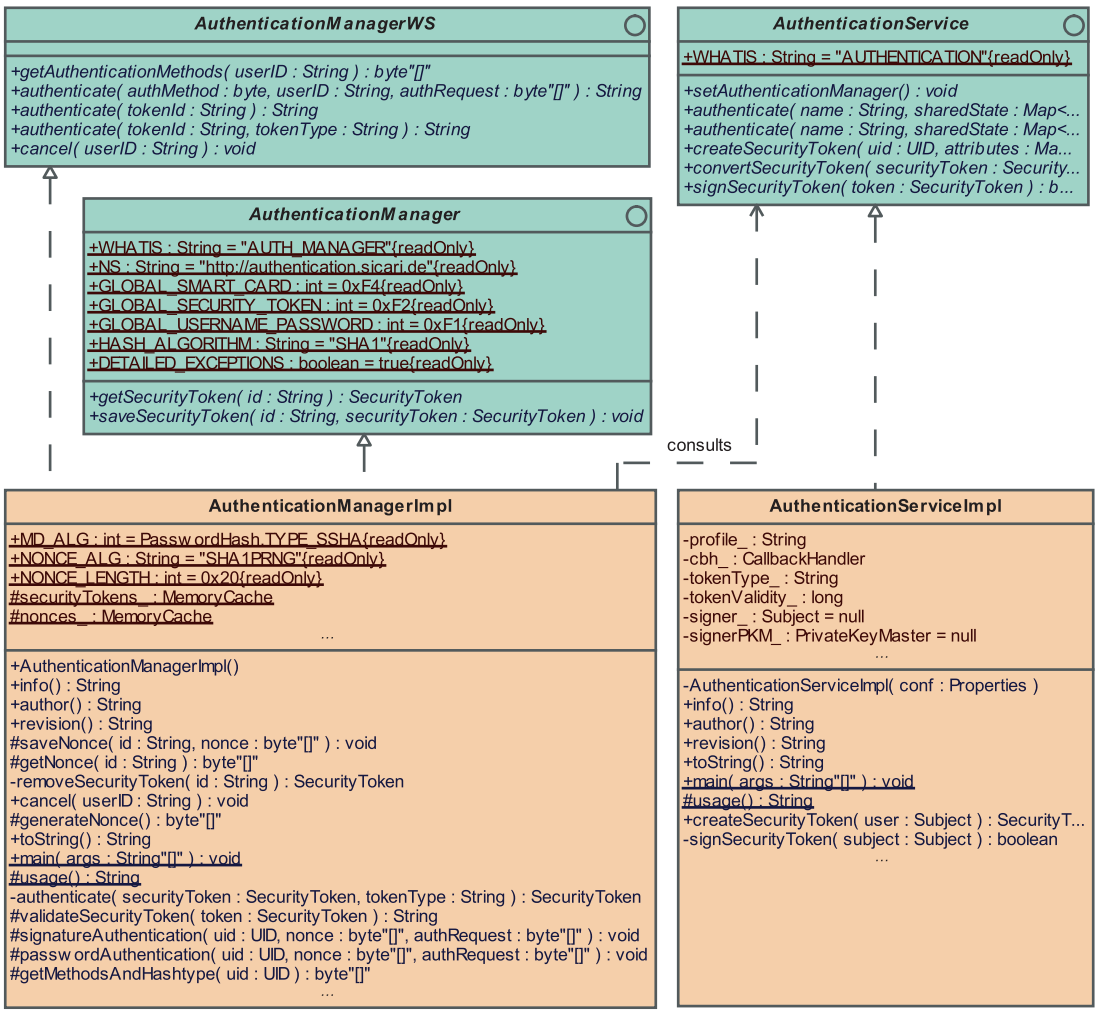

Abbildung 4.13: Klassendiagramm: SicAri-Authentication-Services

– `String authenticate(String tokenId, String tokenType)`:
Mit dieser Methode kann ein ST in den angegebenen Typ konvertiert werden. Diese Methode muss ggf. verwendet werden, wenn für einen Benutzer ein ASN.1-kodiertes ST ausgestellt wurde und der Benutzer einen Dienst nutzen möchte der nur SAML-kodierte STs akzeptiert. Auch hier ist der Rückgabeparameter eine ID, mit der das konvertierte ST identifiziert werden kann.

4.2.1.5 AuthenticationManagerHandler

Der `AuthenticationManagerHandler` (AMH) ist eine wesentliche Komponente bei der globalen Authentifikation von Benutzern. Er stellt eine Implementierung eines Axis-Handler (vgl. Kapitel 3.4.3.1) dar und ist für die Realisierung der auf Web Services basierenden Benutzerauthentifikation unverzichtbar.
Der AMH erbt seine Funktionalitäten von der Klasse `SecurityTokenHandler`. Diese Klasse wird in Kapitel 4.2.2.1 ausführlich vorgestellt.

Der AMH wird jeweils in den Request Flow und Response Flow, der globalen Handlerkette der Axis-Client-Engine, publiziert. In die Axis-Server-Engine wird der Handler ebenfalls in den Request Flow- und Response Flow veröffentlicht, jedoch nicht in der globalen Handlerkette, sondern hier in der servicespezifischen Handlerkette (vgl. Abb. 4.14). Der AMH bildet daher mit dem Provider, der den Authentifikationsmanager-WS aufruft, eine Targeted Chain (siehe Kapitel 3.4.3.1).
Die Veröffentlichung des AMH in die globale Handlerkette der Axis-Client-Engine hat zur Folge, dass alle WS-Nachrichten diesen Handler passieren. Der AMH soll jedoch nur die Nachrichten verarbeiten, die an den Authentifikationsmanager gerichtet sind. Innerhalb der Handler wird daher eine Filterung der Nachrichten vorgenommen. Diese basiert weitestgehend auf XPath-Ausdrücken die über eine Konfigurationsdatei (vgl. den folgenden Abschnitt: Konfiguration des AMH) definiert werden können. Damit wird es möglich diese Filterung zu erweitert, ohne dass Anpassungen an dem Quellcode vorgenommen werden müssen.
Die Konfigurationsdatei definiert für diesen Fall einen XPath-Ausdruck, der alle Nachrichten identifiziert, die nicht den Namensraum des Authentifikationsmanager enthält. Diese Nachrichten werden dann von dem AMH ignoriert. Verschlüsselte Nachrichten an den AMH können auf diese Weise jedoch nicht identifiziert werden. Die Nachrichtenfilterung wurde aus diesem Grund für diesen speziellen Fall erweitert. Anhand der Methode `getTargetService()` und dem Attribut `Call.WSDL_PORT_NAME` des `MessageContext`-Objekts kann überprüft werden, ob eine Nachricht an den Authentifikationsmanager gerichtet ist.
Falls eine Nachricht in dem Request-Flow auf die dargestellte Weise gefiltert wurde, wird im zugehörigen `MessageContext`-Objekt ein Attribut gesetzt, das es erlaubt, die zugehörige Antwortnachricht zu identifizieren und damit zu filtern. Dies ist notwendig, da die Antwortnachrichten teilweise nicht über den Namensraum identifiziert werden können.

Die bei der Kommunikation mit dem Authentifikationsmanager ausgetauschten WS-Nachrichten werden an vier Punkten (vgl. Abb. 4.14) durch den AMH verarbeitet. An diesen Punkten übernimmt der AMH verschiedene Aufgaben:

1. **Axis-Client-Engine** Request-Flow:
   i. Integration des ST A in den SOAP-Header
   ii. Verschlüsseln des ST A
   iii. Signatur des SOAP-Body
   iv. Verschlüsseln des SOAP-Body
2. **Axis-Server-Engine** Request-Flow:
   i. Validierung der SSL/TLS-Session
   ii. Entschlüsseln des SOAP-Body
   iii. Überprüfung der Signatur des SOAP-Body
   iv. Extraktion des ST A aus dem SOAP-Header
   v. Entschlüsseln des ST A
   vi. Validierung des ST A
   vii. Übergabe des ST an den Authentifikationsmanager

3. **Axis-Server-Engine** Response-Flow:
   i. Signatur des SOAP-Body
   ii. Verschlüsseln des SOAP-Body
   iii. Integration des ST B in den SOAP-Header
   iv. Verschlüsseln des ST B
4. **Axis-Client-Engine** Response-Flow:
   i. Entschlüsseln des SOAP-Body
   ii. Überprüfung der Signatur des SOAP-Body
   iii. Extraktion des ST aus dem SOAP-Header
   iv. Entschlüsseln des ST B
   v. Speichern des ST B

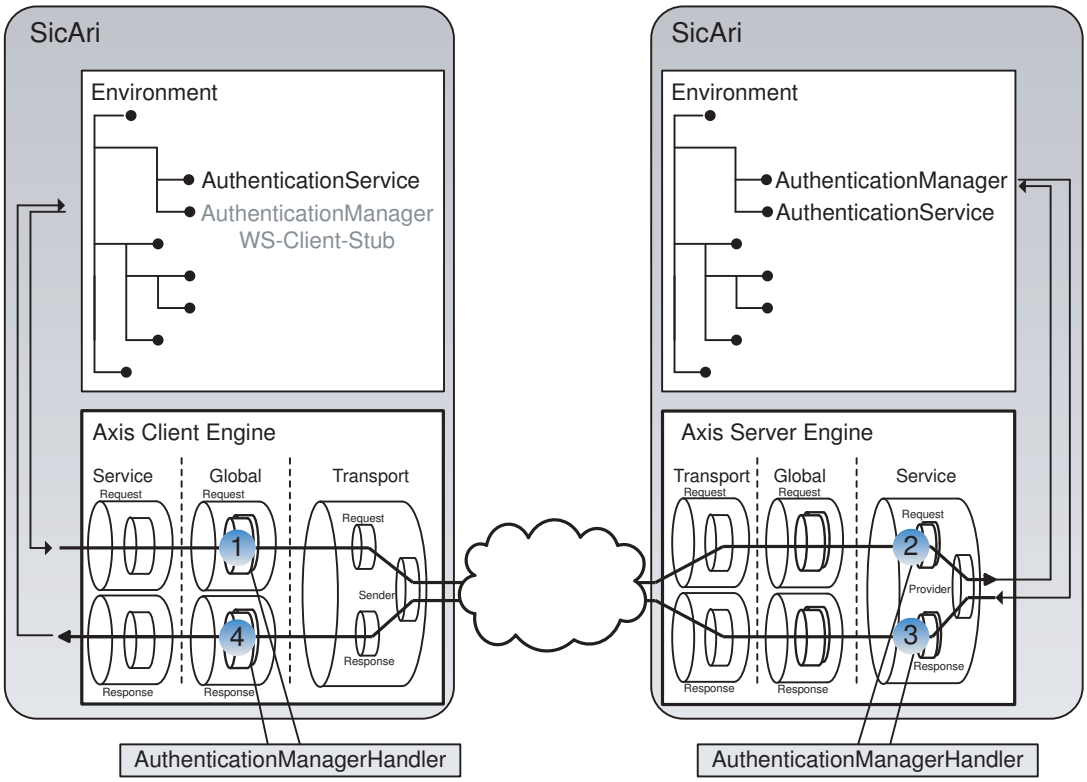

Abbildung 4.14: Der AuthenticationManagerHandler im Authentifikationsprozess

Die wichtigste Aufgabe des AMH ist die Integration bzw. Extraktion von Security Tokens in die bzw. aus den zu verarbeitenden SOAP-Nachrichten. In Kapitel 4.2.3 wird erklärt, warum dies von einem Axis-Handler durchgeführt werden muss, und das ST nicht als Rückgabeparameter eines WS-Aufrufs übertragen werden kann. Der AMH muss dies ggf. an vier Punkten im Nachrichtenfluss vornehmen.

Falls die Nutzung von bestimmten Diensten der Plattform ggf. ein anderes Format des ST voraussetzt als das ursprünglich ausgestellte, kann über den Authentifikationsmanager ein ST mit dem benötigten Format erstellt werden. An Punkt 1 muss der AMH in diesem Fall das aktuelle ST in die SOAP-Nachricht integrieren. Der AMH extrahiert dieses ST an Punkt 2 wieder und übergibt es für die Konvertierung an den Authentifikationsmanager. Nachdem dieser das ST konvertiert hat, wird es an Punkt 3 in die Antwortnachricht integriert. Schließlich wird es auf Client-Seite an Punkt 4 aus der SOAP-Nachricht extrahiert und kann dann weiter verarbeitet werden.
Bei der initialen Benutzerauthentifikation wird kein ST von der lokalen Plattform zur entfernten übertragen. Der AMH integiert das vom Authentifikationsmanager ausgestellte ST nur an Punkt 3. An Punkt 4 wird es wieder extrahiert und mit dem `Subject` des Benutzers verknüpft.
Der AMH kann die Security Tokens, die in die Nachricht integriert werden müssen, ggf. verschlüsseln. In Abhängigkeit der Punkte an denen dies notwendig ist, muss festgestellt werden, für wen die Verschlüsselung des ST erfolgen soll. Hierfür wird jeweils der öffentliche Schlüssel der empfangenden Instanz benötigt. In Kapitel 4.2.6 wird dargestellt, wie dieser bezogen werden kann.
Falls für die Verbindung nicht das SSL/TLS-Protokoll verwendet wird, können für die Sicherung der Vertraulichkeit und Integrität alternativ WS-Security-Mechanismen aktiviert werden. Dazu werden an den einzelnen Punkten die eintreffenden SOAP-Nachrichten ver- bzw. entschlüsselt und außerdem eine Signatur erstellt bzw. diese validiert. Bei der Verschlüsselung einer Nachricht muss festgestellt werden, für wen diese Verschlüsselung erfolgen soll. In Kapitel 4.2.4 wird dargestellt, wie die Verschlüsselung und Signatur mittels WS-Security-Mechanismen realisiert wird.
Eine weitere wichtige Aufgabe des AMH ist die Validierung einer aufgebauten SSL/TLS-Sitzung an Punkt 3. In Kapitel 4.2.5 wird hierauf näher eingegangen.
Die Verarbeitung der Nachrichten innerhalb des AMH wird weitestgehend durch XPath gesteuert. Mittels XPath-Ausdrücken wird beispielsweise überprüft, ob eine Nachricht entschlüsselt werden muss oder welcher ST-Typ die Nachricht enthält. Auf diese XPath-gesteuerte Verarbeitung wird in Kapitel 4.2.7 ausführlich eingegangen.

Konfiguration des AMH Die dargestellten Aufgaben werden nur dann ausgeführt, wenn sie aktiviert werden. Der AMH kann deshalb über eine Datei flexibel konfiguriert werden. Dazu wurde die Datei `ws-am-handler.conf` angelegt. Der AMH kennt dabei die folgenden Konfigurationsparameter.

- `authentication.level`: Mit diesem Parameter wird die Authentifikationsstufe festgelegt. Der AMH unterstützt dabei die im Folgenden beschriebenen Stufen. Die Authentifikationsstufen 2 und 3 werden nur innerhalb des Handler verarbeitet, wenn dieser in der Axis-Server-Engine publiziert ist. Damit erfolgt die Validierung der SSL/TLS-Sitzung nur auf Server-Seite des Handler. Standardmäßig ist die Authentifikationsstufe 2 ausgewählt.

  – Stufe 3: Wird die Authentifikationsstufe 3 ausgewählt, so wird der Zugriff auf den Authentifikationsmanager nur dann gestattet, wenn die Verbindung mittels SSL/TLS verschlüsselt ist und die Identität der anfragenden Plattform erfolgreich über die Option Client-Authentifikation des SSL/TLS-Protokolls festgestellt werden konnte. Zudem wird ggf. ein an den Authentifikationsmanager zu sendendes ST bzw. ein von diesem Dienst ausgestelltes ST in die Anfrage- bzw. Antwortnachricht integriert.

  - Stufe 2: Wird die Authentifikationsstufe 2 ausgewählt, so wird der Zugriff auf den Authentifikationsmanager nur dann gestattet, wenn die Verbindung mittels SSL/TLS verschlüsselt ist. Zudem wird ggf. ein an den Authentifikationsmanager zu sendendes ST bzw. ein von diesem Dienst ausgestelltes ST in die Anfrage- bzw. Antwortnachricht integriert.
  - Stufe 1: Wird diese Stufe ausgewählt, wird nicht überprüft, ob die Verbindung mittels SSL/TLS abgesichert ist. Gegebenenfalls wird ein an den Authentifikationsmanager zu sendendes ST bzw. ein von diesem Dienst ausgestelltes ST in die Anfrage- bzw. Antwortnachricht integriert.
  - Stufe 0: Wird die Authentifikationsstufe 0 ausgewählt, so erfolgen keine Überprüfungen im AMH. Diese Stufe ist lediglich für Testzwecke vorhanden und sollte im laufenden Betrieb nicht verwendet werden.

- `token.encryption`: Dieser Parameter legt fest, ob ein ST vor der Übertragung verschlüsselt werden soll. Durch Angabe der Werte 0 bzw. 1 kann die Verschlüsselung aktiviert oder deaktiviert werden. Bei aktivierter Verschlüsselung wird diese für die Plattform, an die das ST gesendet werden soll, vorgenommen. Standardmäßig ist die Verschlüsselung deaktiviert.

- `request.message.encryption`: Dieser Konfigurationsparameter definiert, ob die Anfragenachricht verschlüsselt werden soll. Durch Angabe der Werte 1 bzw. 0 kann die Verschlüsselung aktiviert bzw. deaktiviert werden. Standardmäßig ist die Verschlüsselung deaktiviert.

- `request.message.signature`: Dieser Konfigurationsparameter legt fest, ob die Anfragenachricht signiert werden soll. Durch Angabe der Werte 1 bzw. 0 wird die Signatur der Nachricht vorgenommen bzw. nicht vorgenommen. Standardmäßig erfolgt keine Signatur der Anfragenachricht.

- `response.message.encryption`: Dieser Konfigurationsparameter legt fest, ob die Antwortnachricht verschlüsselt werden soll. Durch Angabe der Werte 1 bzw. 0 kann die Verschlüsselung aktiviert bzw. deaktiviert werden. Standardmäßig ist die Verschlüsselung deaktiviert.

- `response.message.signature`: Dieser Konfigurationsparameter legt fest, ob die Antwortnachricht signiert werden soll. Durch Angabe der Werte 1 bzw. 0 wird die Signatur der Nachricht vorgenommen bzw. nicht vorgenommen. Standardmäßig erfolgt keine Signatur der Antwortnachricht.

- `xpath.filter.#`: Über diesen Parameter können mehrere XPath-Ausdrücke angegeben werden, die festlegen, welche Nachrichten der Handler nicht bearbeiten soll. Jeder XPath-Ausdruck wird dabei durch eine fortlaufende Nummer nach dem Parameternamen festgelegt. Die XPath-Ausdrücke werden in Oder-verknüpfter Form verarbeitet. Standardmäßig wird nur ein XPath-Ausdruck festgelegt:

  ```
  xpath.filter.1 =
  not(//*[namespace-uri()='http://authentication.sicari.de'])
  ```

  Dieser Wert legt fest, dass alle Nachrichten nicht verarbeitet werden sollen, die nicht den Namensraum `http://authentication.sicari.de` enthalten. Dies bedeutet, dass der AMH nur auf solche Nachrichten reagiert, die an den Authentifikationsmanager gerichtet sind.

Dieser Aufbau ermöglicht die flexible Erweiterung der Nachrichtenfilterung ohne Anpassung des Quellcodes.

Ein Beispiel für diese Konfigurationsdatei ist im Anhang in Kapitel E.1.3 aufgeführt.

4.2.2 Sicherheitsenforcement

Nachdem sich ein Benutzer erfolgreich an der Plattform angemeldet und damit authentifiziert hat, kann er im Rahmen seiner Sicherheitsrichtlinien die Dienste der Plattform nutzen. Möchte er nun einen Dienst nutzen, der nicht auf der lokalen Plattform verfügbar ist, aber auf einer entfernten Plattform existiert, so geschieht dies für den Benutzer in transparenter Weise. Allenfalls durch einen verzögerten Aufruf merkt der Benutzer, dass er einen entfernten Dienst nutzt.
Für die Durchsetzung der Sicherheitsrichtlinien auf der entfernten Plattform, muss diese natürlich feststellen können, um welchen Benutzer es sich handelt. Der Benutzer muss sich deshalb ebenfalls gegenüber der entfernten Plattform authentifizieren. Dies geschieht implizit durch die Übertragung des ST an die entfernte Plattform. Das ST wird auf der entfernten Plattform dazu verwendet, für den aufrufenden Benutzer einen neuen Sicherheitskontext aufzubauen. Die Übertragung des ST von einer lokalen zu einer entfernten Plattform wird durch einen speziellen Axis-Handler vorgenommen, der nun im Folgenden vorgestellt wird.

4.2.2.1 SecurityTokenHandler

Der `SecurityTokenHandler` (STH) wird zur Realisierung der impliziten Authentifikation von Benutzern auf entfernten SicAri-Plattform eingesetzt. Damit wird mit diesem Handler das SSO innerhalb der SicAri-Infrastruktur umgesetzt.
Der STH ist wie der AMH ebenfalls eine Implementierung eines Axis-Handler. Abbildung 4.15 zeigt das Klassendiagramm des STH, auf das in den weiteren Ausführungen dieses Kapitels Bezug genommen wird.
Da nicht abzusehen ist, welche Dienste ein Benutzer auf einer entfernten Plattform nutzt, müssen alle WS-Nachrichten an diese Dienste den STH passieren. Nur so kann der STH für die Authentifikation des Benutzers auf der entfernten Plattform sorgen. Wie bereits in Kapitel 3.4.3.1 vorgestellt, bietet Axis verschiedene Handlerketten an, um einen Handler in der Axis-Engine zu platzieren. Für Service-übergreifende Verarbeitungen existiert die Global-Handlerkette. In diese Handlerkette wird deshalb der STH jeweils in den Request- und Response-Flow der Axis-Server- und Axis-Client-Engine platziert.
Der STH darf jedoch Nachrichten an die UDDI-Registry und den Axis-AdminService-WS nicht verarbeiten. Außerdem darf er auch nicht die Nachrichten, die an den Authentifikationsmanager gerichtet sind, verarbeiten. Um diese Nachrichten zu ignorieren, arbeitet der Authentifikationsmanager ebenfalls mit einer XPath-basierten Filterung. Innerhalb der zugehörigen Konfigurationsdatei (vgl. den folgenden Abschnitt: Konfiguration des STH) werden entsprechende XPath-Ausdrücke definiert. Die Nachrichtenfilterung arbeitet ansonsten genauso wie im Kapitel 4.2.1.5 beschrieben. So werden beispielsweise die verschlüsselten Nachrichten an den Authentifikationsmanager ebenfalls über die Attribute des `MessageContext`-Objekts identifiziert.
In Abhängigkeit der Position des Handler in der Axis-Engine (vgl. Abbildung 4.16) übernimmt dieser die im Folgenden vorgestellten Aufgaben. Diese Aufgaben werden innerhalb der Methode `invoke(MessageContext)` ausgeführt. Wie bereits in Kapitel 3.4.3.1 dargestellt, kann über die Methoden `isClient()` und `getPastPivot()` des `MessageContext` festgestellt werden, an welchem Punkt die SOAP-Nachricht den Handler passiert. Anhand dieser Position wird entschieden ,welche Aufgaben ausgeführt werden müssen:

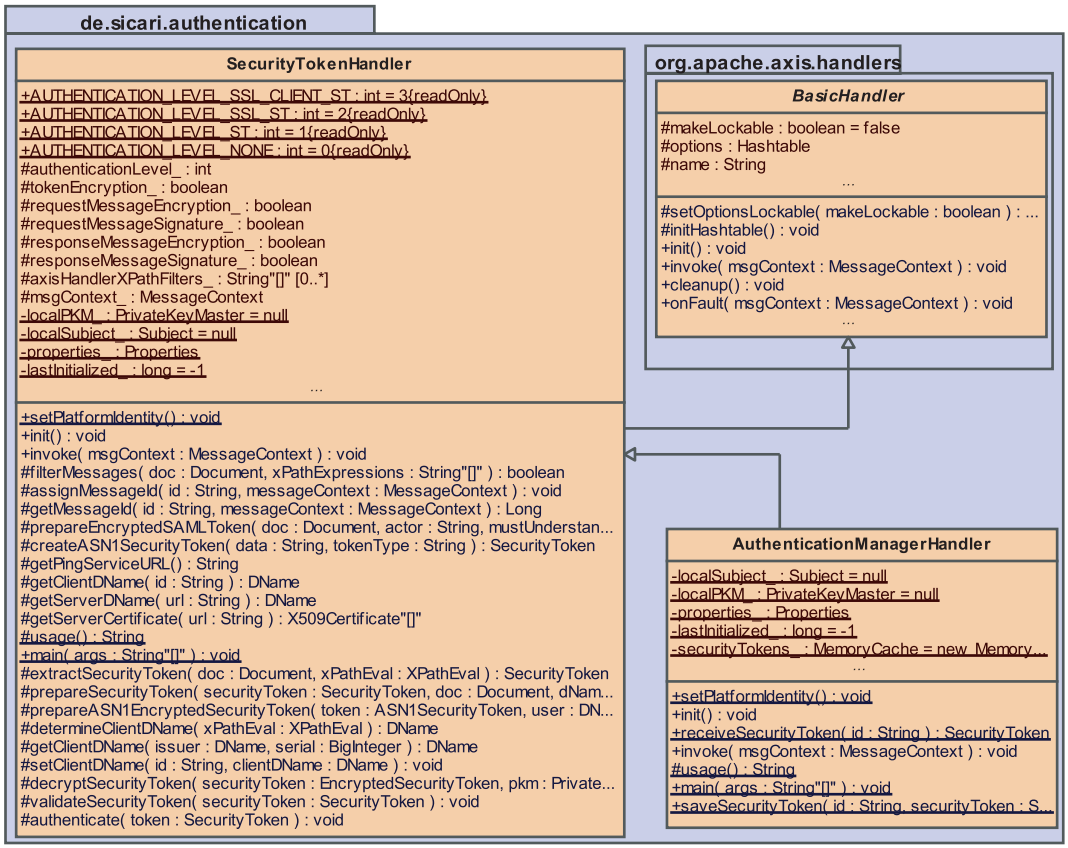

Abbildung 4.15: Klassendiagramm: Axis-Handler

1. **Axis-Client-Engine** Request-Flow:
   i. Signatur des SOAP-Body
   ii. Verschlüsseln des SOAP-Body
   iii. Zugriff auf das ST
   iv. Verschlüsseln des ST
   v. Integration des ST in den SOAP-Header
2. **Axis-Server-Engine** Request-Flow:
   i. Validierung der SSL/TLS-Session
   ii. Entschlüsseln des SOAP-Body
   iii. Überprüfung der Signatur des SOAP-Body
   iv. Extraktion des ST aus dem SOAP-Header
   v. Entschlüsseln des ST
   vi. Validierung des ST
   vii. Aufbau des neuen Sicherheitskontext

3. **Axis-Server-Engine** Response-Flow:
   i. Signatur des SOAP-Body
   ii. Verschlüsseln des SOAP-Body
4. **Axis-Client-Engine** Response-Flow:
   i. Entschlüsseln des SOAP-Body
   ii. Überprüfung der Signatur des SOAP-Body

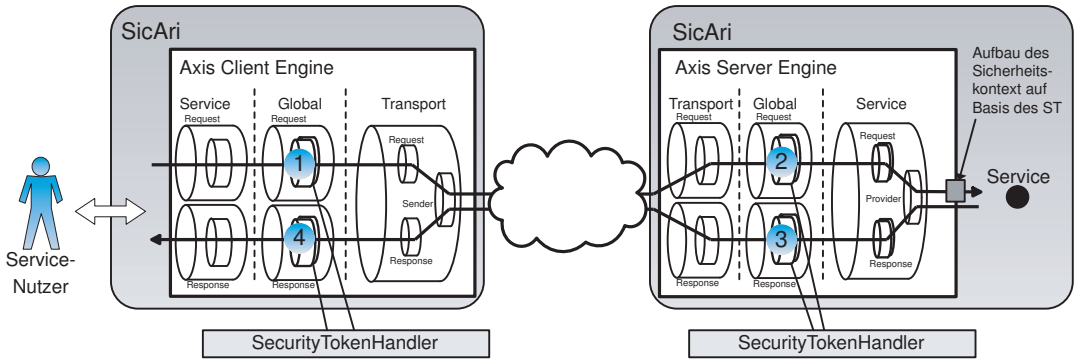

Abbildung 4.16: Der `SecurityTokenHandler` im SSO-Prozess

Der Benutzer muss sich für die Nutzung eines Dienstes bei der entfernten Plattform zunächst authentifizieren. Diese Authentifikation wird für den Benutzer in transparenter Weise durch den STH durchgeführt.
Dazu muss an Punkt 1 das ST des Benutzers ausgelesen werden und in die SOAP-Nachricht, die an den gewünschten Dienst gerichtet ist, eingebettet werden. Im STH wird über den Aufruf `Subject.getSubject(AccessController.getContext())` zunächst das aktuelle `Subject` ausgelesen. Wie bereits erwähnt, enthält das `Subject` das ST in Form eines privaten Credentials. Über die Methode `getPrivateCredentials(SecurityToken.class)` der Klasse `Subject` kann auf das ST zugegriffen werden.
Nachdem das ST verfügbar ist, wird es in den Header der SOAP-Nachricht integriert. Dazu wird die Methode `prepareSecurityToken(...)` des STH aufgerufen. Der STH kann wie auch der AMH das Security Token, das in die Nachricht integriert werden muss, ggf. verschlüsseln. Auch hier wird für die Verschlüsselung der öffentliche Schlüssel der empfangenden Instanz benötigt (vgl. Kapitel 4.2.6). Innerhalb dieser Methode wird das Security Token deshalb ggf. mit Hilfe der Methode `prepareASN1EncryptedSecurityToken(...)` bzw. der Methode `prepareEncryptedSAMLToken(...)` verschlüsselt. Danach wird die Methode `embed(Document)` des ST aufgerufen. Diese Methode fügt das ST als Kindelement eines neuen SOAP-Header in das übergebene `Document`-Objekt ein.
An Punkt 2 überprüft der STH ggf. die bestehende SSL/TLS-Verbindung. In Kapitel 4.2.5 wird hierauf näher eingegangen. Konnte die SSL/TLS-Verbindung erfolgreich validiert werden, extrahiert der STH das Security Token aus der SOAP-Nachricht. Dazu ruft der STH die Methode `extractSecurityToken(...)` auf. Innerhalb dieser Methode wird zunächst anhand von XPath-Ausdrücken festgestellt, um welches ST es sich handelt. Danach wird anhand dem in der

Nachricht enthaltenen SAML-Assertion ein SAML-ST bzw. den Base64-kodierten Daten ein ASN.1-ST (`createASN1SecurityToken(...)`) erstellt.
Danach überprüft der STH das ST innerhalb der Methode `valdiateSecurityToken`. Dazu ruft er die Methoden `verify`, `validate` und `isValid` nacheinander auf. Dadurch wird die Signatur des ST verifiziert, überprüft ob der ausstellenden Instanz vertraut werden kann und schließlich der Gültigkeitszeitraum des ST festgestellt.
Auf Basis des ST wird auf der entfernten Plattform schließlich ein neuer Sicherheitskontext erstellt und der Zugriff auf den aufgerufenen Dienst gewährt.
Der Handler ermöglicht es zudem, die Sicherung der Vertraulichkeit und Integrität über WS-Security Mechanismen zu gewährleisten. Dazu werden an den Punkten 1 und 3 die eintreffenden SOAP-Nachrichten verschlüsselt und außerdem eine Signatur erstellt. An den Punkten 2 und 4 werden die Nachrichten dann wieder entschlüsselt und die Signatur überprüft. Bei der Verschlüsselung einer Nachricht muss festgestellt werden, für wen diese Verschlüsselung erfolgen soll. In Kapitel 4.2.4 wird dargestellt, wie die Verschlüsselung und Signatur mittels WS-Security-Mechanismen realisiert wird.
Die Verarbeitung der Nachrichten innerhalb des STH wird wie auch schon beim AMH weitestgehend durch XPath-Ausdrücke gesteuert. Auf diese XPath-gesteuerte Verarbeitung wird in Kapitel 4.2.7 ausführlich eingegangen.

Konfiguration des STH Genauso wie beim AMH können die einzelnen Aufgaben des STH mit Hilfe einer Konfigurationsdatei aktiviert und deaktiviert werden. Zur Konfiguration dieses STH wurde die Datei `ws-st-handler.conf` angelegt. Der STH versteht dabei die gleichen Konfigurationsparameter wie der `AuthenticationManagerHandler`. Daher wird hier nur auf die geänderten Standardwerte eingegangen und zudem die veränderte Bedeutung der Authentifikationsstufen für diesen Handler verdeutlicht.

- `authentication.level`: Mit diesem Parameter wird die Authentifikationsstufe festgelegt. Der STH unterstützt dabei die im Folgenden beschriebenen Stufen. Die Authentifikationsstufen 2 und 3 werden nur innerhalb des Handler verarbeitet, wenn dieser in der Axis-Server-Engine publiziert ist. Damit erfolgt die Validierung der SSL/TLS-Sitzung nur auf Server-Seite des Handler. Standardmäßig ist die Authentifikationsstufe 2 ausgewählt.

  – Stufe 3: Wird die Authentifikationsstufe 3 ausgewählt, so wird der Zugriff auf einen dahinterliegenden Dienst nur dann gestattet, wenn die Kommunikationsverbindung mittels SSL/TLS verschlüsselt ist und die Identität der anfragenden Plattform erfolgreich mittels SSL/TLS-Client-Authentifikation festgestellt werden konnte. Zudem ist für den Zugriff ein ST (des Benutzers, der die Anfrage erstellt hat) erforderlich. Dieses muss erfolgreich validiert werden können und dessen Gültigkeitszeitraum darf nicht abgelaufen sein.
  – Stufe 2: Wird die Authentifikationsstufe 2 ausgewählt, so wird der Zugriff auf einen dahinterliegenden Dienst nur dann gestattet, wenn die Kommunikationsverbindung mittels SSL/TLS verschlüsselt ist. Zudem ist für den Zugriff ein ST erforderlich. Dieses muss erfolgreich validiert werden können und dessen Gültigkeitszeitraum darf nicht abgelaufen sein.
  – Stufe 1: Wird die Authentifikationsstufe 1 ausgewählt, so wird der Zugriff auf einen dahinterliegenden Dienst nur dann gestattet, wenn die Nachricht ein ST enthält, dieses erfolgreich validiert werden konnte und dessen Gültigkeitszeitraum nicht abgelaufen ist.

  - Stufe 0: Wird die Authentifikationsstufe 0 ausgewählt, so erfolgen keine Überprüfungen im `AuthenticationManagerHandler`. Diese Stufe ist lediglich für Testzwecke vorhanden und sollte im laufenden Betrieb nicht verwendet werden.

- `xpath.filter.#`: Über diesen Parameter können mehrere XPath-Ausdrücke angegeben werden, die festlegen, welche Nachrichten der STH nicht bearbeiten soll. Jeder XPath-Ausdruck wird dabei durch eine fortlaufende Nummer nach dem Parameternamen festgelegt. Die XPath-Ausdrücke werden in Oder-verknüpfter Form verarbeitet.

  Standardmäßig sind die folgenden XPath-Ausdrücke festgelegt.

  - `xpath.filter.1 = //*[starts-with(namespace-uri(),'urn:uddi-org:api')]`
    Durch Angabe dieses Werts werden alle Nachrichten ignoriert, die Namensräume an beliebiger Stelle der SOAP-Nachricht enthalten, die mit `urn:uddi-org:api` beginnen. Dieser XPath-Ausdruck ist deshalb so aufgebaut, um auch Nachrichten der UDDI-Versionen 2 und 3 filtern zu können. Diese besitzen die Namensräume `urn:uddi-org:api_v2` und `urn:uddi-org:api_v3`. Alle Nachrichten die diese Namensräume enthalten, sind an die UDDI-Registry gerichtet bzw. kommen von der UDDI-Registry und dürfen nicht verarbeitet werden.
  - `xpath.filter.2 = //*[namespace-uri()='http://authentication.sicari.de']`
    Durch Angabe dieses Werts werden alle Nachrichten ignoriert, die den Namensraum `http://authentication.sicari.de` an beliebiger Stelle der SOAP-Nachricht enthalten. Alle Nachrichten die diesen Namensraum enthalten, sind an den Authentifikationsmanager gerichtet bzw. kommen von diesem Dienst und dürfen nicht verarbeitet werden.
  - `xpath.filter.3 = //*[namespace-uri()='http://xml.apache.org/axis/wsdd/']`
    Durch Angabe dieses Werts werden alle Nachrichten ignoriert, die den Namensraum `http://xml.apache.org/axis/wsdd/` an einer beliebigen Stelle der Nachricht enthalten. Diese Nachrichten sind an den AdminService-WS von Axis (vgl. Kapitel 3.4.3.1) gerichtet bzw. kommen von diesem Dienst und dürfen nicht verarbeitet werden.

  Dieser Aufbau ermöglicht die flexible Erweiterung der Nachrichtenfilterung ohne Anpassung des Quellcodes.

Ein Beispiel für diese Konfigurationsdatei ist im Anhang in Kapitel E.1.2 aufgeführt.

4.2.3 ST Übertragung

Security Tokens müssen im Fall des SSO von einer lokalen zu einer entfernten Plattform bzw. im Fall der globalen Benutzerauthentifikation vom Authentifikationsmanager zur lokalen Plattform bzw. zu einem Endgerät übertragen werden.
Im Fall des SSO muss das ST in die SOAP-Nachricht durch Hinzufügen eines neuen SOAP-Header integriert werden. Nur so kann das ST für den Benutzer in transparenter Weise zur entfernten Plattform übertragen werden.

Bei der globalen Benutzerauthentifikation wäre es prinzipiell auch denkbar, das ST als Rückgabeparameter der vom Authentifikationsmanager bereitgestellten `authenticate`-Methoden zu definieren. Damit wären die Security Tokens als Teil der Antwortnachricht im SOAP-Body übertragen worden.
Für diese Umsetzung muss jedoch das ST in einen entsprechenden XML-Datentyp umgewandelt werden. Hierfür wäre es notwendig gewesen, zwei spezielle Komponenten zu implementieren. Zunächst einen Serialisierer, der die Java-Datentypen, aus dem das ST aufgebaut ist, in XML-Datentypen umwandelt und außerdem einen Deserialisierer, der die XML-Datentypen wieder in Java-Datentypen zurückwandelt. Für die Umsetzung dieser auch als *Marshalling* bzw. *Unmarshalling* bekannten Technik, wäre die Nutzung der `encode()`-Methode des ST sinnvoll gewesen. Damit wäre das ST auf den Datentyp `byte[]` abgebildet worden. Dieser Datentyp kann mit Hilfe der von Axis angebotenen Standard-*Serializer* leicht in XML abgebildet werden bzw. mit einem Deserializer von XML wieder in den Java-Datentyp gewandelt werden.
Für den speziellen Fall des SAML-ST sollte jedoch auf eine Konformität mit anderen Plattformen geachtet werden. Da eine SAML-ST normalerweise im SOAP-Header übertragen wird, sollte dies auch innerhalb von SicAri so umgesetzt werden. Für eine einheitliche Behandlung aller Security Tokens wurde dies deshalb auch für die anderen Typen beibehalten. Außerdem konnten so die implementieren Funktionen des STH im AMH wiederverwendet werden.
Aus diesem Grund werden als Rückgabe- und Übergabeparemeter der `authenticate`-Methoden des Authentifikationsmanagers entsprechende IDs verwendet, mit denen das zu übertragende ST identifiziert werden kann. Der AMH kann anhand dieser ID, die er aus der entsprechenden SOAP-Nachricht ausliest, das ST vom Authentifikationsmanager abrufen und dann in den SOAP-Header der Nachricht integrieren. Ein ST, das vom Authentifikationsmanager ausgestellt wurde, wird auf Client-Seite im AMH aus der SOAP-Nachricht extrahiert und zwischengespeichert. Das entsprechende Login-Module des JAAS-Framework ruft dann das ST vom AMH über die als Rückgabeparameter erhaltene ID ab.

4.2.4 SOAP-Body Verschlüsselung und Signatur

Durch die vorgestellten Konfigurationsparameter der implementierten Handler kann für Nachrichten, die diese passieren, aktiviert werden, ob diese verschlüsselt und mit einer Signatur versehen werden sollen. Diese Parameter definieren also auf Sender-Seite, ob eine Verschlüsselung oder Signatur erfolgen soll. Auf Empfänger-Seite haben diese Parameter keine Auswirkungen, d. h. falls eine Nachricht verschlüsselt ist oder eine Signatur enthält, so wird versucht, diese zu entschlüsseln bzw. die Signatur zu überprüfen.
Die Konfigurationsparameter erlauben es außerdem, die Vertraulichkeit und Integrität der Nachrichten auch ohne Anwendung von SSL/TLS sicherzustellen. Durch Auswahl der Authentifikationsstufe 1 in Kombination mit SOAP-Body-Verschlüsselung und -Signatur kann gewissermaßen eine fünfte Authentifikationsstufe realisiert werden. In welchem Anwendungsfall dies sinnvoll sein kann, muss die Praxis zeigen.
Für die Verschlüsselung von Nachrichten wird der öffentliche Schlüssel des Empfängers der zu chiffrierenden Nachricht benötigt. Wie dieser identifiziert werden kann, wird im Kapitel 4.2.6 dargestellt.
Die Ver- und Entschlüsselung bzw. die Erzeugung und Validierung einer Signatur wird in den Handler durch die Klasse `SOAPSecurity` (vgl. Abbildung 4.17) realisiert. Diese verwendet intern die Implementierung der WS-Security-Spezifikation *WSS4J*[2]. Die Klasse nutzt eine Reihe von Konstanten für die Initialisierung der Signatur- und Verschlüsselungsalgorithmen. Diese

[2] `http://ws.apache.org/wss4j`

sind in der Schnittstelle `SecurityConstants` definiert. Für die Durchführung der Ver- und Entschlüsselung, bzw. der Erzeugung und Validierung einer Signatur, werden die entsprechenden öffentlichen und privaten Zertifikate bzw. Schlüssel benötigt. Diese werden in Form von so genannten *Crypto-Wrappern*, die mit der Klasse `CryptoCreator` erstellt werden, dem WSS4J-Framework für die Durchführung der einzelnen Aktionen zur Verfügung gestellt.

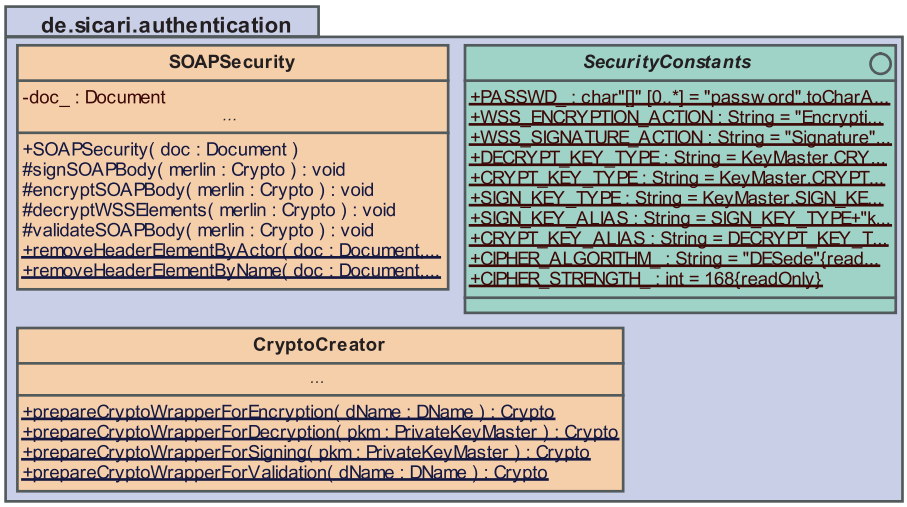

Abbildung 4.17: Klassendiagramm: SOAPSecurity

4.2.5 Validierung der SSL/TLS-Verbindung

Die Validierung einer SSL/TLS-Verbindung wird innerhalb der Handler STH und AMH mit Hilfe der Klasse `SSLSessionVerifier` realisiert. Bei einer eintreffenden Nachricht auf Axis-Server-Seite kann über das `MessageContex`-Objekt auf die bestehende SSL/TLS-Sitzung zugegriffen werden.
Hierfür musste die Implementierung des HTTPS-Server von SicAri erweitert werden. Beim Aufbau einer SSL/TLS-Sitzung wird jetzt im zugehörigen HTTP-`Request`-Objekt eine Referenz auf das `SSLSession`-Objekt gespeichert. Im entsprechenden Handler ist es dadurch möglich, über das `MessageContext`-Objekt auf das HTTP-`Request`-Objekt zuzugreifen, um damit schließlich an das zugehörige `SSLSession`-Objekt zu gelangen.
Das `SSLSession`-Objekt repräsentiert die aufgebaute SSL/TLS-Sitzung und wird an die Klasse `SSLSessionVerifier` (vgl. Abb. 4.18) zur Validierung übergeben. Diese Klasse prüft das bei der SSL/TLS-Sitzung angeforderte Client-Zertifikat. Dazu wird zunächst kontrolliert, ob das Zertifikat einen Plattform-Administrator identifiziert. Danach wird das Zertifikat mit Hilfe des Schlüsselmanagers auf Vertrauenswürdigkeit überprüft.

4.2.6 Identifikation der öffentlichen Chiffrierungsschlüssel

Für die Verschlüsselung von STs bzw. SOAP-Nachrichten werden die öffentlichen Schlüssel der Empfänger benötigt. Der Empfänger kann sowohl eine Plattform sein, die einen Dienst bereitstellt, als auch die Plattform von der die Anfrage an einen Dienst stammt und an die die

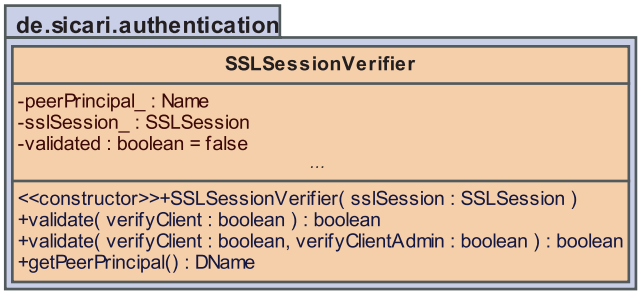

Abbildung 4.18: Klassendiagramm: SSLSessionVerifier

Antwort gesendet werden soll. Die erste Plattform fungiert in diesem Fall als Server und die zweite als Client:

Zugriff auf den öffentlichen Server-Schlüssel Da die Entschlüsselung auf der Plattform erfolgen muss, auf der der angefragte Dienst existiert, wird die Verschlüsselung für diese Plattform durchgeführt. Dazu wird der öffentliche Schlüssel dieser Plattform benötigt. Da jede Plattform durch den Plattform-Administrator gestartet wird, muss hier also auf den öffentlichen Schlüssel des Plattform-Administrators (vgl. Kapitel 3.5.4.1) zugegriffen werden. Zunächst muss also das passende Zertifikat aus der Identitätsdatenbank gelesen werden. Hierfür wird zur Identifikation der so genannte `DName` des Administrators benötigt. Der `DName` repräsentiert einen eindeutigen Benutzernamen (*distinguished name*) innerhalb der SicAri-Infrastruktur.
Die SicAri-Plattform bietet einen speziellen Dienst zur Identifikation einer Plattform an. Der so genannte `PingService` dient dazu, kurzzeitig eine SSL/TLS-Verbindung zu einer Plattform aufzubauen, um auf das Zertifikat des Servers und damit der Plattform zugreifen zu können. Das Zertifikat enthält den `DName` innerhalb des `Subject`-Feld des Zertifikats. Somit kann also der `DName` identifiziert werden.
Zu welcher Plattform über den `PingService` eine Verbindung hergestellt werden soll, kann anhand des Attributs `transport.url` des `MessageContext` festgestellt werden. Dieses Attribut enthält die Zieladresse, an die eine Nachricht gerichtet ist. Über dieses Attribut kann sowohl die IP-Adresse als auch der entsprechende Port des `PingService` ermittelt werden. Der Port ergibt sich dabei aus der Summe von 50000 und der Nummer der SicAri-Instanz. Die Nummer der SicAri-Plattforminstanz kann anhand des Ports, der im Attribut `transport.url` enthalten ist, ermittelt werden. Für die erste SicAri-Plattforminstanz würde dieser entweder im Fall einer HTTP-Verbindung 8081 bzw. einer HTTPs-Verbindung 8881 lauten. Die letzte Ziffer in der Portangabe ergibt die Nummer der Plattforminstanz. Der `PingService` würde also auf der ersten SicAri-Plattforminstanz unter dem Port 50001 laufen.
Da bei der Weiterentwicklung der Plattform oft mehrere Instanzen auf einem PC gestartet werden, und die Belegung eines Netzwerkports durch mehrere Dienste nicht funktioniert, werden die verschiedenen Plattforminstanzen mit unterschiedlichen Konfigurationen gestartet, die jeweils eigene Ports für die einzelnen Netzwerkdienste definieren. Nach Abschluss der Entwicklung und dem Betrieb von mehreren Plattforminstanzen auf unterschiedlichen PCs, wird der `PingService` für alle Plattformen auf einem festgelegten Port betrieben. Für die beschriebene Identifizierung des `DName` wird dann nur noch die IP-Adresse der entsprechenden Plattform benötigt.

Zugriff auf den öffentlichen Client-Schlüssel Auf Server-Seite gibt es keine Möglichkeit, anhand der im `MessageContext` vorhandenen Parameter festzustellen, von wem eine ankommende Nachricht stammt. Momentan lässt sich nur die IP-Adresse feststellen, die anhand eines Attributs im HTTP-`Request`-Objekts des `MessageContext` gespeichert ist. Ähnlich wie beim Zugriff auf die öffentlichen Server-Schlüssel müsste nun der Port des `PingService` ermittelt werden. Derzeit ist der Port dieses Diensts jedoch noch nicht eindeutig festgelegt (vgl. vorheriger Abschnitt).
Innerhalb des Handler ist die Verschlüsselung der Antwortnachricht deshalb nur dann möglich, wenn außerdem auch die Signatur der Anfragenachricht aktiviert ist. Anhand dem in der Signatur enthaltenen Zertifikats, kann der `DName` des Erstellers der Signatur identifiziert werden. Dazu wird aus dem Zertifikat der Aussteller und die Seriennummer ausgelesen und damit über die Identitätsdatenbank das zugehörige Zertifikat abgerufen. Der in diesem Zertifkat enthaltene `DName` identifiziert den Administrator der Plattform, von der die Anfragenachricht stammt und damit auch die Instanz, für die die Verschlüsselung der Antwortnachricht erfolgen soll. Anhand des `DName` kann nun über die Identitätsdatenbank auf den benötigten öffentlichen Schlüssel zugegriffen werden.
Der Handler bietet prinzipiell auch die Möglichkeit, den `DName` anhand einer aktivierten SSL- bzw. TLS-Verbindung zu identifizieren. Dazu muss die Authentifikationsstufe 3 ausgewählt werden. Über das nun durch die Client-Authentifikation angeforderte Zertifikat ist es möglich, den `DName` der anfragenden Plattform festzustellen. In der Praxis ist es jedoch fraglich, ob eine Kombination von SSL/TLS und SOAP-Body Verschlüsselung/Signatur durch die WS-Security Mechanismen sinnvoll ist. Die Aktivierung beider Techniken würde keinen Mehrwert bringen und die Leistungsfähigkeit unnötig verschlechtern.

4.2.7 XPath gesteuerte Verarbeitung

Die Verarbeitung der SOAP-Nachrichten in den implementierten Axis-Handler wird weitgehend anhand von XPath-Ausdrücken gesteuert. Mit Hilfe dieser Ausdrücke wird beispielsweise überprüft, ob eine SOAP-Nachricht verschlüsselt ist oder welches ST sie enthält.
Ein XPath-Ausdruck definiert einen Pfad in einem XML-Baum. Im Handler wird überprüft, ob ein XPath-Ausdruck in einem `Document`-Objekt existiert. Dieses `Document`-Objekt repräsentiert die Baumstruktur der zu verarbeitenden SOAP-Nachricht (vgl. Kapitel 2.4.1 & 2.4.2).
Beim Einlesen einer eintreffenden SOAP-Nachricht wird im Handler über das Axis-Framework die Nachricht in ein `Document`-Objekt überführt. Bei dem Parsen dieser Nachricht werden die Namensräume des XML-Dokuments berücksichtigt. Beim Aufbau eines XPath-Ausdrucks müssen deshalb diese Namensräume ebenfalls beachtet werden, d. h. dass das jeweilige Prefix des aktuellen Namensraums bei der Pfadangabe angegeben werden muss.
Wie ein Prefix eines Namensraums auszusehen hat, wird von der jeweiligen Spezifikation jedoch nicht definiert. Deshalb kann beispielsweise für den Namensraum der SOAP-Spezifikation das Prefix `soapenv`, `SOAP-ENV` oder auch eine ganz andere Zeichenkette verwendet werden.
Von Implementierung zu Implementierung können sich Prefixes daher unterscheiden. Beim Aufbau eines XPath-Ausdrucks muss jedoch, zur eindeutigen Identifikation eines Pfades im XML-Baum, das Prefix bekannt sein. Die implementierten Handler nutzen deshalb für die Angabe der Prefixes entsprechende Platzhalter, die in der Klasse `XPathEval` definiert sind (vgl. Abbildung 4.19).
Deshalb wird in einem Handler zunächst ein so genannter Namensraum-Kontext erstellt. Dieser wird innerhalb von SicAri durch die Klasse `DocumentNamespaceContext` aufgebaut. Beim Einlesen eines XML-Dokuments durch diese Klasse werden alle bestehenden Namensräu-

me und definierten Prefixes in einer Liste gespeichert. Dadurch kann überprüft werden, welcher Namensraum welches Prefix besitzt und welches Prefix zu welchem Namensraum gehört.
Für die Evaluierung von XPath-Ausdrücken wird die Klasse `XPathEval` (vgl. Abbildung 4.19) verwendet. Diese erzeugt zunächst anhand des `Document`-Objekts den Namensraum-Kontext. Danach wird für alle relevanten Namensräume überprüft, welche Prefixes diese besitzen und diese Werte gespeichert. Soll nun ein XPath-Ausdruck evaluiert werden, werden die angegebenen Platzhalter durch die konkret vorhandenen Prefixes ersetzt und danach überprüft, ob der XPath-Ausdruck einen gültigen Pfad im XML-Dokument darstellt.

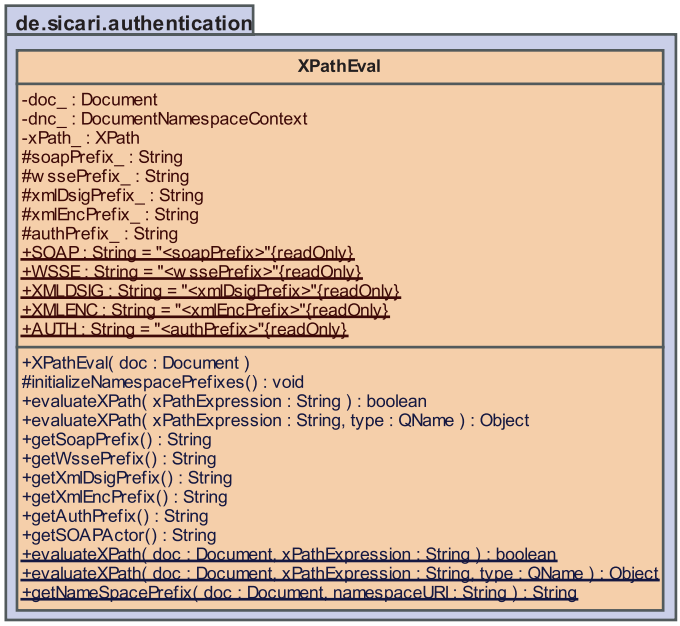

Abbildung 4.19: Klassendiagramm: XPathEval

Die Klasse `XPathEval` berücksichtigt derzeit die Namensräume der Spezifikation SOAP 1.1, SOAP 1.2, XMLdsig, XMLenc, WSS sowie den Namensraum des Authentifikationsmanager-Web Service.
Diese Implementierung ermöglicht damit die Evaluierung von XPath-Ausdrücken, ohne dass für den Entwickler bekannt sein muss, welches Prefix im konkreten Fall verwendet wird.

Im einzelnen werden die folgenden XPath-Ausdrücke zur Steuerung der Verarbeitung innerhalb der Handler verwendet.

- Erkennen einer WS-Security-Signatur oder -Verschlüsselung
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/`
  `<wssePrefix>:Security[@<soapPrefix>:actor='Signature']` (SOAP 1.1)
  bzw.
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/`
  `<wssePrefix>:Security[@<soapPrefix>:role='Encryption']` (SOAP 1.2)

- SOAP-Faults:
  Im Fehlerfall kann ein SOAP-Fault den Handler durchlaufen. Diese Nachrichten dürfen

nicht weiter verarbeitet werden. Anhand des folgenden XPath-Ausdrucks kann überprüft werden, ob es sich um einen SOAP-Fault handelt.
`/<soapPrefix>:Envelope/<soapPrefix>:Body/<soapPrefix>:Fault/`

- Ob eine SOAP-Nachricht ein Security Token enthält kann durch die folgenden XPath-Ausdrücke determiniert werden.

  ASN1SecurityToken:
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/`
  `<wssePrefix>:BinarySecurityToken[@ValueType='ASN1SecurityToken']`

  ASN1EncryptedSecurityToken:
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/`
  `<wssePrefix>:BinarySecurityToken[@ValueType='ASN1EncryptedSecurityToken']`

  SAMLSecurityToken:
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/`
  `<wssePrefix>:Security[@<soapPrefix>:role='Assertion']` (SOAP 1.2)

- Zur Feststellung des `DName` des Ausstellers und der Seriennummer eines Zertifikats einer Signatur werden die folgenden XPath-Ausdrücke verwendet.

  DName des Austellers:
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/<wssePrefix>:Security/`
  `<xmlDsigPrefix>:Signature/<xmlDsigPrefix>:KeyInfo/`
  `<wssePrefix>:SecurityTokenReference/<xmlDsigPrefix>:X509Data/`
  `<xmlDsigPrefix>:X509IssuerSerial/`
  `<xmlDsigPrefix>:X509IssuerName/child::text()`

  Seriennummer des Zertifikats:
  `/<soapPrefix>:Envelope/<soapPrefix>:Header/<wssePrefix>:Security/`
  `<xmlDsigPrefix>:Signature/<xmlDsigPrefix>:KeyInfo/`
  `<wssePrefix>:SecurityTokenReference/<xmlDsigPrefix>:X509Data/`
  `<xmlDsigPrefix>:X509IssuerSerial/`
  `<xmlDsigPrefix>:X509SerialNumber/child::text()`

- Wird eine Authentifikationsanfrage an den Authentifikationsmanager gesendet, kann es notwendig sein, dass der `AuthenticationManagerHandler` ein Security Token in die Nachricht einbetten muss. Abbildung 4.20 zeigt beispielhaft den Aufbau der Authentifikationsanfrage einer SOAP-Nachricht. Durch folgenden XPath-Ausdruck kann überprüft werden, welches Security Token in diese Nachricht integriert werden muss. Anhand des XPath-Ausdrucks wird in diesem Fall die ID `sicari1` ausgelesen. Über diese ID kann der AMH das passende ST abrufen und in die Nachricht integrieren.
  `//*/<authPrefix>:authenticate/tokenId/child::text()`

- Hat der Authentifikationsmanager einen Benutzer erfolgreich authentifiziert, so sendet er eine Antwort mit der ID des ausgestellten Security Tokens zurück. Der zugehörige Axis-Handler `AuthenticationManagerHandler` muss in diesem Fall ein Security Token in die Nachricht einbetten. Durch folgenden XPath-Ausdrucks kann auch hier überprüft werden, welches Security Token in die Nachricht integriert werden muss. Der Wert

```
 1:<soapenv:Envelope
 2:  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 3:  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 4:  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 5: <soapenv:Body>
 6:  <ns1:authenticate
 7:    soapenv:encStyle="http://schemas.xmlsoap.org/soap/encoding/"
 8:    xmlns:ns1="http://authentication.sicari.de">
 9:   <tokenId
10:     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
11:     xsi:type="soapenc:string">
12:    sicari1
13:   </tokenId>
14:  </ns1:authenticate>
15: </soapenv:Body>
16:</soapenv:Envelope>
```

Abbildung 4.20: Anfragenachricht zur Authentifikation

`authenticateReturn` identifiziert den Rückgabeparameter der `authenticate`-Methode. Abbildung 4.21 zeigt den Aufbau einer typischen Antwortnachricht des Authentifikationsmanagers.

```
//*/authenticateReturn/child::text()
```

```
 1:<soapenv:Envelope
 2:  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 3:  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 4:  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 5: <soapenv:Body>
 6:  <ns1:authenticateResponse
 7:    soapenv:encStyle="http://schemas.xmlsoap.org/soap/encoding/"
 8:    xmlns:ns1="http://authentication.sicari.de">
 9:   <authenticateReturn
10:     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
11:     xsi:type="soapenc:string">
12:    sicari1
13:   </authenticateReturn>
14:  </ns1:authenticateResponse>
15: </soapenv:Body>
16:</soapenv:Envelope>
```

Abbildung 4.21: Antwortnachricht zur Authentifikation

4.2.8 Fehlerbehandlung

Während der Implementierungen wurden zwei neue `Exception`-Klassen eingeführt:

- `SecurityTokenException`: Dieser Exception-Typ tritt auf, wenn ein Problem beim Verarbeiten eines ST auftritt. Das kann beispielsweise ein Problem beim Erstellen oder Verifizieren einer Signatur oder Einbetten des ST in ein `Document`-Objekt sein.

- `AuthenticationException`: Dieser Exception-Typ wird in den implementierten Axis-Handler STH und AMH immer dann erzeugt, wenn ein Problem bei der Authentifikation auftritt. Dies kann z. B. auftreten, wenn ein verschlüsseltes ST nicht dechiffriert werden kann.

  Außerdem wird dieser Exception-Typ erzeugt, wenn im Authentifikationsmanager die Authentifikation fehlschlägt.

Alle Exceptions, die in den Axis-Handler auftreten, werden innerhalb der `invoke(...)`-Methode abgefangen und in einen `AxisFault` konvertiert. Die Klasse `AxisFault` ist eine Spezialisierung der Klasse `java.rmi.RemoteException` und eine Implementierung eines SOAP-Fault (vgl. Kapitel 2.4.3.2). Dieser wird schließlich über das Axis-Framework an die anfragende Instanz weitergeleitet. Die anfragende Instanz wird so über einen auftretenden Fehler informiert und kann ggf. darauf reagieren.

4.2.9 Mobile Endgeräte

In diesem Kapitel soll nun untersucht werden, welche der in Kapitel 4.1.3 dargestellten Authentifikationszenarien mit der Java ME umgesetzt werden können und wie ein mobiles Endgerät die Dienste der SicAri-Plattform nutzen kann. Dazu wird zunächst ein kurzer Überblick über die optionalen APIs der Java ME, die für die Realisierung der Authentifikationszenarien relevant sind, gegeben. Die Kommunikation mit der Plattform kann über den SicAri-WS-Gateway realisiert werden. In Kapitel 4.2.9.2 werden deshalb verschiedene Frameworks vorgestellt, mit denen der Zugriff auf die SicAri-Web Services umgesetzt werden kann. Danach werden die eingesetzten Entwicklungswerkzeuge und deren Konfiguration vorgestellt. Schließlich wird evaluiert, welche Authentifikationszenarien realisiert werden können.

4.2.9.1 Optionale APIs

Wie bereits in Kapitel 2.9.1 erwähnt, besteht die Java ME aus optionalen Paketen. Für den Zugriff auf die SicAri-Plattform sind hier besonders die Pakete *Java ME Web Services* (JSR 172), *Security and Trust Services APIs* (JSR 177) sowie *Security* (JSR 219) zu betrachten:

Java ME Web Services-Spezifikation Das optionale Paket *Java ME Web Services* (JSR 172) ermöglicht es, innerhalb einer Java ME-Anwendung auf einen Web Service zu zugreifen. Das Paket besteht aus zwei unabhängigen Paketen, die für den Aufruf eines entfernten Service sowie für das Parsen von XML-Nachrichten mittels SAX (vgl. Kapitel 2.4.2.1) verwendet werden. Das Paket kann sowohl in der CLDC als auch in der CDC verwendet werden und bietet Unterstützung für die Spezifikationen SOAP 1.1, WSDL 1.1, XML 1.0 und XML Schema. Eine Unterstützung für UDDI und damit für das dynamische Finden von Web Services bietet das Paket nicht.
Um einen Web Service mit den in diesem Paket definierten APIs aufzurufen, muss zunächst mit einem entsprechendem Programm aus dem WSDL-Dokument ein WS-Stub generiert werden. Der Stub kümmert sich um das Kodieren und Dekodieren der Übergabe- und Rückgabeparameter und kommuniziert mit der JSR 172-API um den WS aufzurufen.
Die generierten Java-Sourcen müssen deshalb zunächst auf dem Gerät installiert werden, um einen gewünschten WS nutzen zu können. Diese Vorgehensweise ist für den praktischen Einsatz sehr umständlich und ist nur sinnvoll einzusetzen, wenn immer der gleiche Web Service genutzt werden möchte.

Security and Trust Services APIs Die *Security and Trust Services APIs* kurz SATSA sind ein optionales Paket, das auf dem Profil MIDP 2.0 aufbaut. Es ist damit unter der CLDC einsetzbar. Für die Nutzung dieses Pakets muss zusätzlich die *Java Card*-Plattform installiert werden. Diese stellt eine Java-Plattform für Anwendungen dar, die auf Smart Cards laufen.
Das SATSA-Paket besteht aus folgenden Teilen:

- **SATSA-Application Protocol Data Unit (APDU)**: Dieses Paket definiert eine API, mit der der Zugriff auf Smart Cards über das APDU-Protokoll realisiert wird [99].
- **SATSA-Java Card Remote Method Invocation (JCRMI)**: Dieses Paket definiert die Client-API für den Aufruf von Methoden einer angeschlossenen Smart Card [99].
- **SATSA-Public Key Infrastructure (PKI)**: Mit Hilfe der in diesem Paket definierten API können digitale Signaturen durch Aufruf der entsprechenden Methode einer Smart Card erstellt werden. Außerdem enthält das Paket einfache Methoden für das Zertifikatsmanagement [99].
- **SATSA-CRYPTO**: Dieses Paket stellt eine Teilmenge der JCE dar und ermöglicht die Überprüfung von digitalen Signaturen sowie die Erstellung von Verschlüsselungen und Hash-Werten.[99]

Security Das Security Paket besteht aus den folgenden Spezifikationen. Für deren Einsatz wird das *Foundation Profile* vorausgesetzt.

- **Java Secure Socket Extension**: Das *Java Secure Socket Extension (JSSE)*-Framework ist eine optionales Paket der CDC und stellt eine Java-Implementierung des SSL/TLS-Protokolls dar. Das Framework umfasst Funktionen zur Datenverschlüsselung, zur Sicherung der Nachrichtenintegrität sowie zur Authentifikation von Servern und optional Clients. Diese Version umfasst alle Pakete, die auch aus der Java SE 1.4.2 bekannt sind.
- **Java Cryptography Extension**: Das *Java Cryptography Extension (JCE)*-Paket umfasst eine Sammlung von kryptografischen Verschlüsselungsverfahren. Dazu gehören symmetrische und asymmetrische Verschlüsselungsalgorithmen, Verfahren zur Schlüsselerzeugung und Protokolle zum sicheren Schlüsselaustausch.

  Alle diese Funktionen sind im Paket `javax.crpyto` bzw. Unterpaketen davon spezifiziert. Das JCE-Paket hat den gleichen Funktionsumfang wie das JAAS-Paket der Java SE 1.4.2.
- **Java Authentication and Authorization Service**: Innerhalb der CDC kann mit dem optionalen Paket JAAS (JSR 219) ebenfalls das bereits aus der Java SE bekannte Framework zur Authentifikation und Autorisierung von Benutzern eingesetzt werden. Das JAAS-Framework wurde bereits in Kapitel 4.1.1.1 umfassend vorgestellt.

  Die Funktionen des JAAS der CDC besitzen fast den gleichen Umfang wie die JAAS-Pakete der Java SE 1.4.2. Lediglich das Paket `javax.security.auth.kerberos` ist nicht enthalten.

4.2.9.2 WS-Frameworks

Neben der Java ME Web Services-Spezifikation (vgl. Kapitel 4.2.9.1) existieren noch weitere Java-Frameworks, die den Zugriff auf Web Services ermöglichen:

- **kSOAP2**[3] ist eine freie Web Service-Client Bibliothek für die CLDC bzw. die CDC. kSOAP2 erlaubt es in der aktuellen Version nicht, auf die Header einer SOAP-Nachricht zuzugreifen. Für die Umsetzung der verschiedenen Authentifikationszenarien muss jedoch auf ein ST und damit auf den SOAP-Header zugegriffen werden können. Daher wurde das WS-Framework WSOAP eingesetzt.

- **WSOAP** ist ebenfalls eine Client-Bibliothek für den Zugriff auf Web Services. Sie wurde von dem Unternehmen *Wingfoot Software*[4] entwickelt und wird zur freien Nutzung angeboten. Die Bibliothek ist SOAP 1.1 konform und kann sowohl unter der CLDC als auch der CDC eingesetzt werden.

Prinzipiell lassen sich diese Frameworks auch unter der Java SE bzw. Java EE einsetzen. Beide Frameworks nutzen für die Verarbeitung der SOAP-Nachrichten das XML-Framework kXML[5]. kXML implementiert einen so genannten *Pull-Parser* zur Verarbeitung von XML-Dokumenten. Das bedeutet, dass die Anwendung selbst die Steuerung durch das XML-Dokument übernimmt. Dazu ruft sie entsprechende Methoden des Parsers auf, um die verschiedenen Teile des Dokuments zu verarbeiten.
Abbildung 4.22 zeigt den Zugriff auf den Authentifikationsmanager-WS mit der WSOAP-Bibliøthek. Der Quelltext beschreibt wie die Methode `getAuthenticationMethods` dieses WS aufgerufen wird. Zunächst wird eine SOAP-Nachricht erstellt und der Namensraum, der Methodennamen und die entsprechenden Parametern gesetzt. Danach wird mit dem Objekt `HTTPTransport` der Zugriff auf den WS über das HTTP-Protokoll abgewickelt. Dazu wird zunächst die URL des WS angegeben und der Hostname als HTTP-Header spezifiziert. Schließlich wird der WS aufgerufen und die Antwort ausgelesen.

4.2.9.3 Entwicklungswerkzeuge & Konfiguration

Bevor die einzelnen Authentifikationszenarien auf ihre praktische Umsetzbarkeit untersucht werden, wird in diesem Kapitel ein Überblick über die verwendeten Entwicklungswerkzeuge gegeben.

Netbeans IDE Netbeans ist eine Entwicklungsumgebung (*Integrated Development Environment (IDE)*), die die Erstellung von Programmen ermöglicht. Sie ist komplett in Java programmiert und steht als Open Source zur Verfügung. Netbeans ermöglicht die Entwicklung von Programmen in fast jeder Programmiersprache und steht für verschiedene Betriebssysteme zur Verfügung. Netbeans besitzt eine hervorragende Unterstützung für die Programmierung von Anwendungen für die Java ME. Hierfür existieren zwei *Add-on*-Pakete:

- *Netbeans Mobility Pack for CLDC/MIDP 5.5.1*: Mit diesem Add-on können Programme für die CLDC erstellt, getestet und debugged werden. Das Paket erlaubt es außerdem, verschiedene Emulatoren einzubinden und damit die Funktionen auf einem virtuellen Gerät zu testen.

- *Netbeans Mobility Pack for CDC 5.5.1*: Ähnlich wie das Add-on für die CLDC ermöglicht dieses Paket die Entwicklung von Anwendungen für die CDC. Auch hier können weitere Geräte-Emulatoren von Dritt-Anbietern eingebunden werden.

[3] http://ksoap2.sourceforge.net
[4] http://www.wingfoot.com
[5] http://kxml.sourceforge.net

```
/* Prepare the envelope*/
Envelope reqEnv = new Envelope();

reqEnv.setSchema("http://www.w3.org/1999/XMLSchema");
reqEnv.setSchemaInstance("http://www.w3.org/1999/XMLSchema-instance");

reqEnv.setBody("userID", userID);

/* Prepare the call*/
Call call = new Call(reqEnv);
call.setMethodName("getAuthenticationMethods");
call.setTargetObjectURI("http://authentication.sicari.de");

/* Prepare the transport */
HTTPTransport transport =
  new HTTPTransport(
    "https://hostname:8881/webservice/services/security/auth-manager",
    null);
transport.addHeader("host", "hostname");
transport.getResponse(true);

/* Make the call*/
Envelope resEnv = call.invoke(transport);

if (resEnv.getParameterCount() >= 1)
{
  Object res = resEnv.getParameter(0);
  if (res instanceof Base64)
  {
    byte[] authMethods = ((Base64) res).getBytes();
  }
}
```

Abbildung 4.22: Web Service-Zugriff unter Verwendung des WSOAP-Framework

Für die Installation der *Netbeans 5.5.1 IDE* wird ein bereits installiertes *Java Development Kit (JDK)* der Version 5 oder 6 vorausgesetzt. Nach der Installation der IDE müssen noch die zwei Add-on Pakete installiert werden.
Nach der Installation müssen für die CLDC und CDC jeweils verschiedene Einstellungen vorgenommen werden:

- **CLDC**: Für die CLDC muss innerhalb von Netbeans der Java ME Emulator des Wireless Tooklit 2.5.1 (siehe folgender Abschnitt) verfügbar gemacht werden. Dazu muss im Menüpunkt *Tools* der *Java Platform Manager* ausgewählt werden. Danach muss über die Schaltfläche *Add Plattform* eine neue Plattform hinzugefügt werden. Hierfür muss zunächst in der Liste der Eintrag *Java Micro Edition Platform Emulator* ausgewählt werden. Danach muss das Installationsverzeichnis des Wireless Tooklit 2.5 ausgewählt und schließlich die Einstellungen gespeichert werden.

  Für den erfolgreichen Aufbau einer SSL/TLS-Verbindung von dem mobilen Endgerät zur SicAri-Plattform ist es notwendig, dass die CLDC das CA-Zertifikat der SicAri-PKI (vgl.

Kapitel 2.6.4) kennt. Das Zertifikat muss daher dem Emulator bekannt gemacht werden. Dazu muss das *Utilities*-Programm `utilsw.exe` des Wireless Toolkit gestartet werden. Über die Schaltfläche *Manage Certificates* wird das Zertifikatsmanagement gestartet. Nun muss das Zertifikat über die Schaltfläche *Import Certificate* in den Emulator importiert werden. Als *Security Domain* muss beim Importieren noch der Wert *Manufacturer* ausgewählt werden. Das Zertifikat muss dafür im X.509-Format vorliegen.

- **CDC**: Für den erfolgreichen Aufbau einer SSL/TLS-Verbindung von dem mobilen Endgerät zur SicAri-Plattform ist es notwendig, dass die CDC-Plattform das CA-Zertifikat der SicAri-PKI kennt. Dazu muss das CA-Zertifikat in die Datei `cacerts` aufgenommen werden. Die Datei befindet sich im Unterverzeichnis `cdc/CDCTK10/lib/security` des Installationsverzeichnises der Netbeans-Installation. Mit dem mit der Java-Installation verfügbaren Programm `Keytool` kann das entsprechende Zertifikat in diese Datei aufgenommen werden. Das Zertifikat muss dafür im X.509-Format vorliegen

```
keytool -import -trustcacerts -keystore cacerts
  -storepass mynewpwd -alias sicarica -file sicari-ca.cer
```

Zudem muss das WSOAP-Framework eingebunden werden. Dazu muss im Menüpunkt *Tools* der *Library Manager* ausgewählt werden. Über die Schaltfläche *New Library* wird nun eine neue Bibliothek angelegt. Als *Library Name* wird WSOAP angegeben und als *Library Type* muss *Mobile Libraries* festgelegt werden. Schließlich muss noch über die Schaltfläche *Add JAR* die Jar-Datei des WSOAP-Frameworks ausgewählt werden. Beim Erstellen eines CLDC- bzw. CDC- Projekts kann nun in den Projekteinstellungen die WSOAP-Bibliothek eingebunden werden.

SUN Java Wireless Toolkit 2.5.1 for CLDC Das *Java Wireless Toolkit 2.5.1 for CLDC* ist eine Sammlung von Werkzeugen für die Entwicklung von MIDP-Anwendungen. Das Toolkit besteht aus einem Emulator, mit dem die entwickelten Anwendungen getestet werden können. Außerdem aus einer Reihe von verschiedenen Werkzeugen, mit denen beispielsweise ein WS-Stub aus einer WSDL-Beschreibung generiert werden kann.

SUN Java Toolkit 1.0 for CDC Das *Java Toolkit 1.0 for CDC* ist Bestandteil des *Netbeans Mobility Pack for CDC* und stellt verschiedene Werkzeuge für die Entwicklung von Anwendungen für die CDC zur Verfügung.

4.2.9.4 Evaluation der Authentifikationsszenarien

Auf Grundlage der vorgestellten Entwicklungswerkzeuge werden die in Kapitel 4.1.3 spezifizierten Authentifikationsszenarien auf Praktikabilität untersucht. Dazu muss zunächst untersucht werden, welche Authentifikationsmethoden mit der CLDC und der CDC umsetzbar sind.

Authentifikationsmethoden:

- **M1**: Auf beiden Plattformen besteht die Möglichkeit der Berechnung von Hash-Werten. Dies wird durch die Klasse `MessageDigest` ermöglicht. In der CLDC wird diese durch das optionale Paket SATSA-CRYPTO bereitgestellt. In der CDC ist diese Klasse bereits vorhanden, ohne das weitere Pakete installiert werden müssen. Die Authentifikation mit

Benutzername/Passwort ist deshalb sowohl unter der CLDC als auch unter der CDC möglich.

- **M2**: Unter der CLDC ist die Authentifikation mittels eines Soft Token nicht möglich, da die dafür notwendige Funktion der Erstellung von digitalen Signaturen nicht vorhanden ist und auch nicht durch ein optionales Paket nach installiert werden kann. Evtl. wäre es aber möglich, diese Funktion von einer angeschlossenen Smart Card ausführen zu lassen. Durch das verfügbare Paket SATSA ist es möglich, auf die Funktionen einer Smart Card zuzugreifen. Im Rahmen dieser Arbeit konnte dies jedoch aus Zeitgründen nicht weiter untersucht werden.

  Unter der CDC können Signaturen erstellt werden, wenn das Profil FP installiert ist. Danach stehen alle notwendigen Funktionen, die auch aus der Java SE bekannt sind, zur Verfügung. Die Berechnung einer Signatur kann deshalb in gewohnter Weise durchgeführt werden. Abbildung 4.23 zeigt dies anhand eines Quelltextausschnitts. Hier wird die Authentifikationsanfrage mit Hilfe des privaten Schlüssels, der innerhalb eines Soft Token gespeichert ist, berechnet.

```
String pwd = "...";
String ksPath = "file:///...";

KeyStore ks = KeyStore.getInstance("JKS");
InputConnection conn = Connector.open(ksPath, Connector.READ);
ks.load(conn.openInputStream(), pwd.toCharArray());

PrivateKey privateKey = ks.getKey("signkey", pwd.toCharArray());

String sigalg = "SHA1with"
   + privateKey.getAlgorithm();

Signature signature = Signature.getInstance(sigalg);
signature.initSign(privateKey);
signature.update(userID.getBytes());
signature.update(nonce);

byte[] sig = signature.sign();
```

Abbildung 4.23: Berechnung der Soft Token-basierenden Authentifikationsanfrage unter der CDC

Unter die Methode $M2$ fällt auch die SSL/TLS-Client-Authentifikation auf die nachfolgend in diesem Kapitel eingegangen wird.

- **M3**: Das Paket SATSA ermöglicht unter der CLDC Plattform den Zugriff auf eine Smart Card. Ob damit auch die Authentifikation mittels Smard Card möglich ist, konnte im Rahmen dieser Arbeit aus Zeitgründen nicht weiter untersucht werden. Unter der CDC Plattform existieren keine Funktionen, die den Zugriff auf eine Smard Card erlauben.

- **M4**: Für diese Authentifikation ist es zunächst notwendig, dass sich ein Benutzer über sein mobiles Endgerät mit einer der vorherigen Authentifikationsmethoden bei der Plattform anmeldet. Das ausgestellte ST kann dann für eine implizite Authentifikation verwendet werden.

Wird vom Authentifikationsmanager ein ASN.1-ST ausgestellt, so ist es unter der CDC möglich, dieses ST abzuspeichern. Der Quelltextausschnitt in Abbildung 4.24 zeigt den Aufruf der `authenticate`-Methode unter Verwendung des WSOAP-Frameworks. Dabei wird die zuvor berechnete Authentifikationsanfrage als Parameter übergeben. Nach der erfolgreichen Authentifikation wird das ASN.1-ST aus dem SOAP-Header ausgelesen und für weitere Zugriffe gespeichert.

```
/* Prepare the envelope */
Envelope reqEnv = new Envelope();
...
reqEnv.setBody("authMethod", new Byte(method));
reqEnv.setBody("userID", userID);
reqEnv.setBody("authRequest", authRequest);

/* Prepare the call */
Call call = new Call(reqEnv);
call.setMethodName("authenticate");
call.setTargetObjectURI("http://authentication.sicari.de");

/* Prepare the transport */
HTTPTransport transport = new HTTPTransport(
    "https://hostname:8881/webservice/services/security/auth-manager",
    null);
transport.addHeader("host", "hostname");
transport.getResponse(true);

/* Make the call */
Envelope resEnv = call.invoke(transport);

/* Store the header containing the security token */
Enumeration headers = resEnv.getHeader().elements();
while(headers.hasMoreElements())
{
  HeaderEntry headerEntry = (HeaderEntry) headers.nextElement();
  if (headerEntry.getHeaderName().equals("BinarySecurityToken"));
  {
    saveST(headerEntry);
  }
}
```

Abbildung 4.24: Authentifikation unter Verwendung des WSOAP-Framework

Mit einem SAML-ST funktioniert dies jedoch nicht. Das eingesetzte WS-Framework WSOAP ermöglicht es derzeit nicht, auf den kompletten SOAP-Header zuzugreifen, der das SAML-ST beinhaltet. Nach einer entsprechenden Erweiterung dieses Frameworks sollte dies jedoch grundsätzlich möglich sein, sofern die Hardwareressourcen des Geräts dies zulassen.

Bei einem erneuten Zugriff auf einen Dienst kann das ST dann in die Anfragenachricht eingebettet werden. Der Quelltextausschnitt in Abbildung 4.25 zeigt den Aufruf einer WS-Methode. Das gespeicherte ST wird in Zeile 6 als SOAP-Header der Anfragenachricht eingebettet.

```
/*Prepare the envelope*/
Envelope reqEnv = new Envelope();
...

/*Add the stored ST*/
reqEnv.addHeader(getST());

/* Prepare the call*/
Call call = new Call(reqEnv);
call.setMethodName("...");
call.setTargetObjectURI("...");

/* Prepare the transport*/
HTTPTransport transport = new HTTPTransport("...", null);
transport.getResponse(true);

/*Make the call*/
Envelope resEnv = call.invoke(transport);

/* Read response */
for (int i = 0; i < resEnv.getParameterCount(); i++)
{
        Object res = resEnv.getParameter(i);
}
```

Abbildung 4.25: SSO unter Verwendung des WSOAP-Framework

Authentifikationsebenen: Die Authentifikationsebenen legen die möglichen Schnittstellen fest, über die sich ein mobiles Endgerät gegenüber der Plattform authentifizieren kann. Zudem definieren sie auch die Schnittstellen, über die das mobile Endgerät mit der Plattform kommunizieren kann:

- **L4**: Für die Authentifikation bzw. Kommunikation über diese Ebene wird ein Telnet-Client für die Java ME benötigt. Es existieren einige Implementierungen, die jedoch im Rahmen dieser Arbeit aus Zeitgründen nicht mehr getestet werden konnten.

- **L5**: Die Authentifikation über den SicAri-Web Server ist nicht möglich, da die Implementierung des Web Server dies zurzeit nicht unterstützt. Nach einer entsprechenden Erweiterung des Web Servers spricht jedoch grundsätzlich nichts dagegen, diese Authentifikationsebene auch über ein mobiles Endgerät zu nutzen. Der Zugriff auf Webseiten wird innerhalb der Java ME über die Klasse `HttpConntection` ermöglicht.

  Die Kommunikation mit den Diensten der Plattform kann über diese Ebene realisiert werden, wenn die Dienste entsprechende Schnittstellen anbieten.

- **L6**: Die vorgestellten Quelltextausschnitte haben gezeigt, dass die Kommunikation und Authentifikation über den WS-Gateway möglich ist. Die einzelnen Schritte der Authentifikation erfolgen dabei konform zur Protokollspezifikation $AP3$ (vgl. Kapitel 4.1.3.1).

Sicherheit: Ob die Sicherung der Vertraulichkeit und Integrität mit dem SSL/TLS-Protokoll oder den WS-Security-Mechanismen innerhalb der Java ME gewährleistet werden kann, wird im Folgenden dargestellt.

- **SSL/TLS**: Für die Verwendung des SSL/TLS-Protokolls muss innerhalb der CLDC das Profil MIDP verwendet werden. Die CDC ermöglicht dies durch die im optionalen Paket Security (vgl Kapitel 4.2.9.1) vorhandene JSSE-API.

  Beim Aufbau der SSL/TLS-Verbindung über das GCF wird automatisch die Authentizität des Servers anhand des im Protokollverlauf erhaltenen Zertifikats überprüft. Die Client-Anwendung, in diesem Fall also das Java ME-Programm, muss dazu die Instanz, die das Zertifikat ausgestellt hat, kennen. In Kapitel 4.2.9.3 wurde bereits erwähnt, wie die SicAri-CA unter der Konfiguration CLDC bzw. CDC bekannt gemacht werden kann.

  Bei der Überprüfung des Zertifikats wird außerdem kontrolliert, ob der Hostname des Computers zu dem die SSL/TLS-Verbindung aufgebaut wurde, mit dem *Common Name (CN)*-Wert des Eigentümers des Zertifikats übereinstimmt. Für den erfolgreichen Aufbau der SSL/TLS-Verbindung muss deshalb die Plattform mit dem passenden Zertifikat gestartet werden.

  Für den erfolgreichen Aufbau einer SSL/TLS-Verbindung unter der CLDC muss die SicAri-Plattform außerdem mit einem so genannten selbst-signierten (*self signed*) Zertifikat gestartet werden. Ansonsten kommt es zu einer `IOException` bei der Überprüfung des Zertifikats. Ob es sich dabei um einen Fehler in der Implementierung der CLDC handelt, konnte nicht festgestellt werden.

  Mit dem SSL/TLS-Protokoll kann wie bereits erwähnt auch die Authentizität des Clients gegenüber dem Server nachgewiesen werden. Dazu fordert der Server vom Client ein Zertifikat an. Die Implementierung der CLDC unterstützt diese Protokolleigenschaft nicht. Wird die CDC mit dem optionalen Paket JSSE eingesetzt, steht der volle Umfang der SSL/TLS-Implementierung der Java SE 1.4.1 zur Verfügung. Für die Nutzung der Client-Authentifikation muss über folgende Befehle der CDC mitgeteilt werden, welches Zertifikat beim Aufbau der SSL/TLS-Verbindung verwendet werden soll.

  ```
  System.setProperty("javax.net.ssl.keyStore", keystore);
  System.setProperty("javax.net.ssl.keyStorePassword", pwd);
  ```

  Beim Aufbau der Verbindung unter Verwendung des WSOAP-Frameworks werden diese Parameter jedoch ignoriert und die Client-Authentifikation schlägt fehl. Nach intensiver Recherche hat sich gezeigt, dass das WSOAP-Framework für den Aufruf des Web Service die Klasse `HttpConnection` verwendet. Beim automatischen Aufbau der SSL/TLS-Verbindung innerhalb dieser Klasse werden die genannten SSL-Parameter ignoriert. Durch Anpassung der Implementierung und Verwendung der Klasse `HttpURLConnection` anstatt `HttpConnection` konnte dieser Fehler beseitigt werden. Dieser Fehler ist wahrscheinlich auf einen *Bug* in der Implementierung des GCF (vgl. Kapitel 2.9.2) der CDC zurückzuführen. Ein entsprechender *Bug-Request* wurde bei SUN eingereicht.

- **WSS**: Für die Nutzung des WS-Security-Standards wird eine entsprechende Implementierung für die Java ME benötigt. Derzeit existieren jedoch keine Implementierungen für die XMLdsig- und XMLenc-Standards, die für die Realisierung der WSS-Mechanismen

benötig werden. Damit kann der Standard WSS auf einem mobilen Endgerät zur Absicherung der Integrität und Vertraulichkeit der WS-Nachrichten derzeit nicht eingesetzt werden.

Ergebnisse: Für die CLDC ergeben sich damit die in Tabelle 4.5 dargestellten Authentifikationszenarien.

Szenario	Methoden	Ebene	Sicherheit
$S2_{local}$	$M1$,$M4$	$L6$	/
$S3_{local}$	$M1$,$M4$	$L6$	SSL/TLS
$S2_{global}$	$M1$,$M4$	$L6$	/
$S3_{global}$	$M1$,$M4$	$L6$	SSL/TLS
$S1_{remote}$	$M1$,$M4$	$L6$	/
$S2_{remote}$	$M1$,$M4$	$L6$	SSL/TLS

Tabelle 4.5: Authentifikationsszenarien CLDC

Für die CDC ergeben sich die in Tabelle 4.6 dargestellten Authentifikationsszenarien. Die Methode $M2$ kann sowohl über dern Authentifikationsmanager und damit über die Ebene $L6$ als auch über die Option der Client-Authentifikation des SSL/TLS-Protokolls umgesetzt werden.

Szenario	Methoden	Ebene	Sicherheit
$S2_{local}$	$M1$,$M2$,$M4$	$L6$	/
$S3_{local}$	$M1$,$M2$,$M4$	$L6$	SSL/TLS
$S2_{global}$	$M1$,$M2$,$M4$	$L6$	/
$S3_{global}$	$M1$,$M2$,$M4$	$L6$	SSL/TLS
$S1_{remote}$	$M1$,$M2$,$M4$	$L6$	/
$S2_{remote}$	$M1$,$M2$,$M4$	$L6$	SSL/TLS

Tabelle 4.6: Authentifikationsszenarien CDC

4.3 Integration

4.3.1 Service-Bootstrapping

Die Plattform und alle Basis-Dienste von SicAri werden über verschiedene Shell-Skripte (Prefix `rc.`) und Konfigurationsdateien (Endung `.conf`) gestartet. Während der Implementierung dieser Arbeit wurden eine Reihe von Anpassungen vorgenommen, die nun im Folgenden dargestellt werden.
Das Konfigurationsskript `rc.main` dient zum Starten der SicAri-Plattform. In ihr werden eine Reihe von Diensten gestartet und weitere Konfigurationsskripte und -dateien referenziert. Im einzelnen werden diese in der im Folgenden dargestellten Reihenfolge eingelesen. In dieser Liste sind nur die Skripte und Dateien aufgeführt, die im Rahmen dieser Arbeit angepasst werden mussten. In den nächsten Unterkapiteln werden die Funktionen dieser und die vorgenommenen Änderungen vorgestellt. Im Kapitel E des Anhangs sind diese Skripte und Dateien jeweils mit ihrem kompletten Umfang aufgeführt.

1. whatis.conf
2. rc.conf
3. sicari.conf
4. rc.dependencies
5. Innerhalb des Skripts `rc.main` werden nun die Basis-Sicherheitsdienste gestartet. Dazu gehört der Schlüsselmanager `KeyMaster`, der SSL/TLS-Manager `SSLMaster`, der Identitätsmanager `IdentityManager` sowie der `PolicyService`-Dienst. Danach wird der lokale Authentifikationsdienst `AuthenticationService` gestartet. Dieser Dienst kann mit der Datei `authentication.conf` (vgl. Kapitel 4.2.1, Abschnitt Konfiguration) konfiguriert werden. Danach authentifiziert sich der Administrator gegenüber der Plattform. Das Konfigurationsskript `rc.admin` und alle weiteren darin referenzierten Konfigurationsskripte werden nun im Rahmen des Sicherheitskontext des Administrators ausgeführt.
6. rc.admin
   - rc.network
     - webservice.conf

4.3.1.1 whatis.conf

Innerhalb dieser Datei werden eine Reihe von global gültigen Definitionen vorgenommen. Größtenteils sind hier für die verschiedenen SicAri-Dienste Veröffentlichungspfade für das Environment definiert. Anhand dieser Definitionen können diese Dienste über die Klasse `Lookup` gesucht werden. Im Rahmen dieser Arbeit wurde diese Datei um einen Eintrag erweitert. Die Variable `AUTH_MANAGER` definiert den Wert `/security/auth-manager` und damit den Pfad zum Authentifikationsmanager im Environment.

4.3.1.2 rc.conf

Das Konfigurationsskript `rc.conf` definiert eine Reihe von Optionen, die aktiviert oder deaktiviert werden können. Beispielsweise kann über die Option `SSL` die Verwendung von SSL/TLS aktiviert oder deaktiviert werden. Die Option `WSE` legt fest, ob die Web Service-Engine (Axis) gestartet werden soll.
Das Konfigurationsskript wurde um folgende Einträge erweitert.

- `setenv WSE_STH enable`

  Diese Option legt fest, dass der `SecurityTokenHandler` aktiviert werden soll.

- `setenv WSE_STH_CONF_FILE $sicari.etc/ws-st-handler.conf`

  Definiert den Pfad zu der Konfigurationsdatei des `SecurityTokenHandler`.

- `setenv AUTH_MANAGER disable`

  Mit dieser Option wird festgelegt ob der Authentifikationsmanager gestartet werden soll.

- `setenv WSE_AMH enable`

  Diese Option legt fest, dass der `AuthenticationManagerHandler` aktiviert werden soll.

- `setenv WSE_AMH_CONF_FILE $sicari.etc/ws-am-handler.conf`

  Legt den Pfad zu der Konfigurationsdatei des `AuthenticationManagerHandler` fest.

- `setenv AUTHENTICATION_CONF $sicari.etc/authentication.conf`

  Definiert den Pfad zu der Konfigurationsdatei des `AuthenticationService`.

4.3.1.3 sicari.conf

Mit dieser Datei werden benutzerdefinierte Konfigurationen vorgenommen. Dabei werden beispielsweise die Netzwerk-Ports für die verschiedenen Netzwerkdienste (HTTP- & HTTPS-Server, Ping-Service, SSH, ...) festgelegt. Außerdem definieren die Optionen `PF_ADMIN` und `SECURITY_TOKEN` die Identität und den Pfad zu dem Soft Token des Plattform-Administrators. Die Datei wurde um die Option `AUTH_MANAGER` erweitert. Diese überschreibt den Standard-Wert aus der Datei `rc.conf` und definiert ob der Authentifikationsmanager gestartet werden soll.
Durch das Anlegen von mehreren solcher Konfigurationsdateien ist es möglich mehrere SicAri-Plattforminstanzen mit unterschiedlichen Konfigurationen auf einem Computer zu starten.

4.3.1.4 rc.dependencies

Anhand dieses Skripts werden die Abhängigkeiten der verschiedenen SicAri-Dienste aufgelöst. Innerhalb dieses Konfigurationsskripts wird beispielsweise geprüft ob SSL/TLS verfügbar ist. Ist dies nicht der Fall werden abhängige Dienste wie der HTTPS-Server oder der Ping-Service deaktiviert.
Das Skript wurde um mehrere Einträge erweitert. Diese sorgen dafür, dass die Axis-Handler AMH und STH nicht veröffentlicht werden, falls die Web Service-Engine nicht verfügbar sein sollte.

```
if eval ${WSE} ne enable; then
  if eval ${WSE_STH} eq enable; then
    echo "WSE is not available -> WSE_STH disabled."
    setenv WSE_STH disable
  fi
  if eval ${WSE_AMH} eq enable; then
    echo "WSE is not available -> WSE_AMH disabled."
    setenv WSE_AMH disable
  fi
fi
```

4.3.1.5 rc.admin

Innerhalb dieses Skripts werden verschiedene Aufgaben im Sicherheitskontext des Administrators durchgeführt. Dazu gehört die Initialisierung verschiedener Dienste mit dem aktuellen `Subject` und dessen privaten Schlüsselmanager. So erhält der `AuthenticationService` und der SSL/TLS-Manager Zugriff auf die Zertifikate und Schlüssel des Administrators. Diese

Zertifikate werden dann für die Signatur von STs und den Aufbau von SSL/TLS-Verbindungen verwendet.
Danach wird das Skript `rc.network` ausgeführt. Nachdem dieses abgearbeitet ist, wird der private Schlüsselmanager des Administrators an die Axis-Handler `SecurityTokenHandler` und `AuthenticationManagerHandler` übergeben. Dies wird durch den Aufruf der entsprechenden Klasse mit dem Kommandozeilenparameter `-setPlatformIdentity` erreicht. In der `main()`-Methode wird dann die Methode `setPlatformIdentity` aufgerufen. Diese greift über den aktuellen `AccessControlContext` auf das damit verknüpfte `Subject` zu. Über die Methode `getPrivateCredentials(PrivateKeyMaster.class)` des `Subject` kann nun auf den privaten Schlüsselmanager zugegriffen werden. Dieser wird dann in einer privaten statischen Variable dieser Klasse gespeichert. Bei einem erneuten Aufruf der Methode `setPlatformIdentity` wird das aktuelle `Subject` ignoriert. Das `Subject` bzw. der Schlüsselmanager kann also nur einmal gesetzt werden.
Das `rc.admin`-Skript wurde für die beschriebene Funktion um die folgenden Einträge erweitert:

```
${WSE_STH} java de.sicari.authentication.SecurityTokenHandler
  -setPlatformIdentity

${WSE_AMH} java
  de.sicari.authentication.AuthenticationManagerHandler
  -setPlatformIdentity
```

Anschließend wird der Administrator automatisch abgemeldet und der Anmeldeprozess für die regulären Benutzer gestartet.

4.3.1.6 rc.network

Mit diesem Shell-Skript werden die Netzwerk-Dienste gestartet. Neben dem Ping-Service, dem Remote-Shell-Service, dem HTTP- und HTTPS-Server wird hier auch die Web Service-Engine Axis gestartet. Im Rahmen dieser Arbeit wurde dieses Skript um die folgenden Einträge erweitert.

```
echo -n "Authentication manager:"

if eval ${AUTH_MANAGER} eq enable; then
  echo " webservice."
  java de.sicari.authentication.AuthenticationManagerImpl -publish
else
  echo "."
fi

echo -n "Publishing web service handler:"

${WSE_STH} echo -n " st-handler"
${WSE_STH} java de.sicari.authentication.SecurityTokenHandler -deployserver
  -configFile ${WSE_STH_CONF_FILE}
${WSE_STH} java de.sicari.authentication.SecurityTokenHandler -deployclient
  -configFile ${WSE_STH_CONF_FILE}

${WSE_AMH} echo -n " am-handler"
if eval ${AUTH_MANAGER} eq enable; then
  java de.sicari.authentication.AuthenticationManagerHandler -deployserver
    -configFile ${WSE_AMH_CONF_FILE}
```

```
else
  ${WSE_AMH} java de.sicari.authentication.AuthenticationManagerHandler
    -deployclient -configFile ${WSE_AMH_CONF_FILE}
fi

${WSE_AMH} echo "."
```

Zunächst wird hier die Variable `AUTH_MANAGER` überprüft. Ist diese aktiviert, wird der Authentifikationsmanager im Environment in dem Pfad publiziert, der in der Konfigurationsdatei `whatis.conf` zuvor festgelegt wurde.
Danach wird geprüft, ob der `SecurityTokenHandler` aktiviert werden soll. Dieser wird dann sowohl in der Axis-Server-Engine als auch in der Axis-Client-Engine publiziert. Über den Kommandozeilenparameter `-configFile` wird der in dem Skript `rc.conf` mit der Variable `WSE_STH_CONF_FILE` festgelegte Pfad zu der Konfigurationsdatei des Handler angegeben. Ist der Authentifikationsmanager aktiviert, so wird außerdem in der Axis-Server-Engine auch der `AuthenticationManagerHandler` publiziert. Andernfalls wird dieser nur in der Axis-Client-Engine veröffentlicht. Über den Kommandozeilenparameter `-configFile` wird auch hier der Pfad zu der Konfigurationsdatei durch die Variable `WSE_AMH_CONF_FILE` angegeben.

4.3.1.7 webservice.conf

Diese Datei wird als Konfigurationsparameter beim Publizieren des `WebserviceManager` der SicAri-Plattform angegeben. Der `WebserviceManager` ist ein spezieller Dienst, der beim veröffentlichen eines Dienstes im Environment konsultiert wird. Anhand der in dieser Konfigurationsdatei festgelegten Einträge überprüft der `WebserviceManager`, ob ein Dienst auch als Web Service publiziert werden muss. Dies wird anhand des Pfades im Environment festgelegt. Diese Datei wurde um einen Eintrag erweitert der festlegt, dass der Authentifikationsmanager automatisch beim Veröffentlichen im Environment auch als Web Service publiziert wird. Dabei wird außerdem spezifiziert, dass nur das `AuthenticationManagerWS`-Interface als Web Service publiziert werden soll. Dies wird durch das folgende Kommando festgelegt.

```
${WhatIs:AUTH_MANAGER}=
  de.sicari.authentication.AuthenticationManagerWS
```

4.3.1.8 login.conf

Die Datei `login.conf` definiert die Konfigurationsdatei für das JAAS-Framework (vgl. Kapitel 4.1.1.1). Die Java VM besitzt einen Parameter um dem JAAS-Framework den Pfad zu dieser Datei mitzuteilen. Dieser lautet `-Djava.security.auth.login.config`.
Im Rahmen dieser Arbeit wurde der Bereich `default` dieser Datei durch die Verweise auf die Login-Module für die globale Authentifikation ergänzt:

```
default
{
    de.sicari.kernel.security.GlobalPKCS11LoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.pkcs11.conf";
    de.sicari.kernel.security.GlobalJKSLoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.jks.conf";
    de.sicari.kernel.security.GlobalPasswordLoginModule SUFFICIENT;
```

```
    de.sicari.kernel.security.PKCS11LoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.pkcs11.conf";
    de.sicari.kernel.security.JKSLoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.jks.conf";
    de.sicari.kernel.security.PasswordLoginModule REQUISITE;
};
```

4.3.2 WS-Deployment

Das Publizieren von Web Services wird in SicAri über die Klasse `WebserviceService` ermöglicht und gleichzeitig vereinfacht. Diese Klasse bietet dazu zwei Methoden an, um einen Web Service mittels Axis zu veröffentlichen bzw. wieder zu entfernen:

- `deployWebservice`: Unter Angabe des -Namens, der -Klasse und dem Web Service-Interface wird mit dieser Methode ein Web Service automatisch publiziert. Die Angabe des Interface ermöglicht es festzulegen, welche Schnittstellen einer Klassen als Web Service zur Verfügung gestellt werden sollen.
- `undeployWebservice`: Unter Angabe des Web Service-Names kann mit dieser Methode eine zuvor veröffentlichter Web Service wieder entfernt werden.

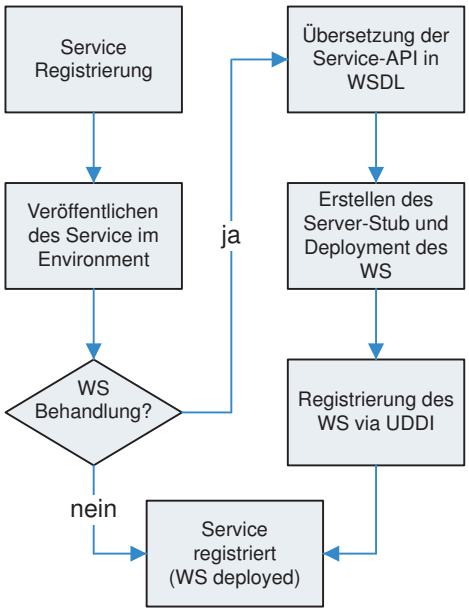

Abbildung 4.26: Service-/WS-Deployment

Normalerweise muss mit Axis zunächst für die Klasse die als Web Service veröffentlicht werden soll, ein so genannter *Web Service Deployment Descriptor (WSDD)* erstellt werden. Dies ist eine Datei mit XML-Inhalt, die den WS beschreibt, verschiedene Aspekte für den SOAP-Endpunkt festlegt und definiert, welche Methoden der Klasse als WS zur Verfügung gestellt werden sollen. Außerdem muss für die Registrierung des WS in der SicAri-UDDI-Registry eine WSDL-Beschreibung generiert werden. Hierfür wird von Axis das Programm `java2wsdl` zur Verfügung gestellt. Anhand des Java-Quellcode generiert dieses Programm automatisch eine WSDL-Beschreibung.

Innerhalb von SicAri werden diese Vorgänge automatisch durch die SicAri-API vorgenommen. Die Klasse `WebserviceLauncher` bildet diese Funktionen außerdem auch auf Shell-Kommandos ab und ermöglicht so das Publizieren von Web Services über die SicAri-Shell.
Abbildung 4.26 zeigt wie innerhalb von SicAri ein Web Service publiziert wird. Der Service wird über die SicAri-API im Environment publiziert. Anhand des angegebenen Pfades wird überprüft, ob der Service auch als Web Service veröffentlicht werden soll (vgl. Kapitel 4.3.1.7). Ist dies der Fall, wird automatisch eine WSDL-Beschreibung generiert, der WS-Server-Stub erstellt und dieser in Axis veröffentlicht. Außerdem wird der WS mittels der WSDL-Beschreibung auch in der UDDI-Registry registriert.

4.3.3 WS-Suche

Soll ein Service verwendet werden, wird im Environment überprüft, ob dieser lokal zur Verfügung steht. Ist dies nicht der Fall, wird in der UDDI-Registry nachgeschaut, ob der entsprechende Service auf einer anderen Plattform vorhanden ist. Wurde ein passender WS gefunden, wird automatisch die WSDL-Beschreibung abgerufen und daraus ein Client-Stub generiert, über den dann der Zugriff auf den WS erfolgt. Gegebenenfalls wird der erstellte Client-Stub auch im lokalen Environment publiziert, so dass der WS bei einem erneuten Aufruf direkt verwendet werden kann. Abbildung 4.27 stellt diesen Vorgang grafisch dar.

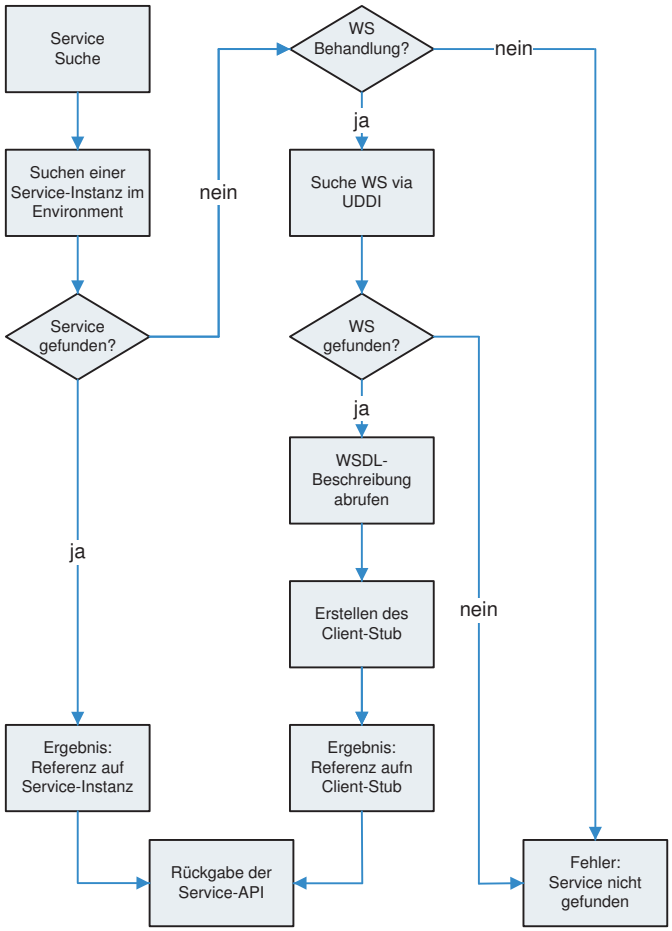

Abbildung 4.27: Service-/WS-Discovery

4.3.4 Handler-Deployment

Auf jeder SicAri-Plattform, auf der die Web Service-Engine Axis gestartet ist, ist es möglich, verschiedene Axis-Handler zu veröffentlichen. Dabei wird zwischen der Axis-Server- und der -Client-Engine unterschieden. Diese besitzen jeweils eine *Transport-*, eine *Global-* und eine *Service*-Handlerkette (Chain), die sich nochmals in die Request- und Response-Seite unterscheiden lassen. Ein Handler kann damit an sechs verschiedenen Punkten einer Engine veröffentlicht werden (vgl. Kapitel 3.4.3.1).
Das Publizieren von Handler wird innerhalb von SicAri ebenfalls wie das Publizieren von Web Services über die Klasse `WebserviceService` ermöglicht. Diese bietet entsprechende Methoden an:

- `deployClientHandler`: Mit dieser Methode wird ein Handler in der Axis-Client-Engine veröffentlicht.
- `undeployClientHandler`: Mit dieser Methode ist es möglich, einen zuvor in der Axis-Client-Engine veröffentlichen Handler wieder zu entfernen.
- `deployServerHandler`: Mit dieser Methode wird ein Handler in der Axis-Server-Engine veröffentlicht.
- `undeployServerHandler`: Mit dieser Methode ist es möglich, einen zuvor in der Axis-Server-Engine veröffentlichen Handler wieder zu entfernen.

Beim Aufruf dieser Methode muss ein Parameter des Typs `HandlerDescriptor` angegeben werden. Die Klasse `HandlerDescriptor` spezifiziert dabei welcher Handler, mit welchen Parametern und in welche Handlerkette der Handler veröffentlicht werden soll.
Die Klasse `WebserviceLauncher` bildet diese Funktionen ebenfalls auf Shell-Kommandos ab und ermöglicht so das Publizieren von Handler auch über die SicAri-Shell.

Kapitel 5

Bedrohungsanalyse

In diesem Kapitel wird eine Bedrohungsanalyse für die Authentifikationsmechanismen der SicAri-Plattform erstellt. Zunächst werden die wichtigsten Grundbegriffe erläutert und dargestellt wie Bedrohungen identifiziert werden können. Zudem wird gezeigt wie Bedrohungen kategorisiert werden können und was typische Bedrohungen sein können.

Anschließend werden die Bedrohungen für die SicAri-Plattform identifiziert und gezeigt wie diese abgewehrt werden. Das Kapitel schließt mit einer Übersicht, die zeigt wie bei der Erstellung einer Risikoanalyse vorgegangen werden muss.

5.1 Methodik

5.1.1 Grundbegriffe: Schwachstelle, Bedrohung und Angriff

Wenn die in Kapitel 2.1.2 vorgestellten Schutzziele beeinträchtigt werden, so geschieht dies durch Ausnutzung von Schwachstellen des Systems. Das System muss sich daher gegen Bedrohungen und Angriffe schützen und so das Risiko einer Beeinträchtigung minimieren. Im Folgenden werden daher die für die weitere Betrachtung wichtigen Begriffe Schwachstelle, Bedrohung, Risiko und Angriff präzisiert.

Definition 5.1 *Schwachstelle und Verwundbarkeit:*
Schwachstellen (Weakness) sind Schwächen eines Systems oder Punkte, an denen das System verwundbar ist. Schwachstellen, über die die Sicherheitsdienste des Systems umgangen, getäuscht oder unautorisiert modifiziert werden können, werden als Verwundbarkeit (Vulnerability) des Systems bezeichnet.[21]

Definition 5.2 *Bedrohung:*
„Eine Bedrohung (Threat) des Systems zielt darauf ab, eine oder mehrere Schwachstellen oder Verwundbarkeiten auszunutzen, um einen Verlust der Datenintegrität, der Informationsvertraulichkeit oder der Verfügbarkeit zu erreichen oder um die Authentizität von Subjekten zu gefährden.“[21]

Definition 5.3 *Risiko:*
Unter der Wahrscheinlichkeit des Eintritts eines Schadensereignisses und der Höhe des potenziellen Schadens, der dadurch hervorgerufen werden kann, wird das Risiko (Risk) einer Bedrohung verstanden.[21]

Definition 5.4 *Angriff:*
Als Angriff (Attack) werden unautorisierte Zugriffe bzw. Zugriffsversuche auf ein System verstanden. Passive Angriffe zielen auf den Verlust der Vertraulichkeit ab, die durch nicht autorisierte Informationsgewinnung entstehen können. Als aktive Angriffe werden dagegen unautorisierte Modifikationen von Datenobjekten bezeichnet. Sie richten sich gegen die Datenintegrität oder Verfügbarkeit eines Systems.[21]

Das System ist bedroht, sobald vorhandene Schwachstellen ausgenutzt werden. Durch Bedrohungen können ganz allgemein Schäden, wie beispielsweise der Verlust von Informationen, Wissen oder auch Vermögen, entstehen. In der Informationstechnik kann ein Schaden konkret den Verlust der Datenintegrität, der Informationsvertraulichkeit oder der Verfügbarkeit bedeuten.
Eine mögliche Klassifizierung von Bedrohungen bzw. Gefährdungen ist in Abbildung 5.1 dargestellt. Diese Klassifizierung orientiert sich an dem Gefährdungskatalog des IT-Grundschutzhandbuchs [34].

Abbildung 5.1: Klassifikation von Gefährdungsfaktoren

Ob eine Schwachstelle für ein IT-System bedrohlich ist, hängt sehr davon ab, wie hoch das Risiko für eine entsprechende Schwachstelle anzusehen ist.
Die Einschätzung des Risikos ist sehr wichtig und hilft Unternehmen, ihre IT-Systeme erfolgreich abzusichern. Das Augenmerk sollte auf Schwachstellen mit dem höchsten Risiko sowie entstehenden Aufwand und Kosten gerichtet werden. Die Bedrohungsanalyse und die Risiko-

analyse sind bewährte Mittel, um Bedrohungen und deren Risiken einschätzen zu können und um so den Schutzbedarf des Systems festzustellen.
Konnte das Risiko nicht soweit eingeschränkt werden, dass ein Schadensereignis eintritt, so liegt ein Angriff vor. Bei passiven Angriffen liegt die Schwierigkeit sehr oft in der Erkennung eines Angriffs bzw. eines Angriffversuchs. Ob Informationen, die über einen Kommunikationskanal versendet werden, abgehört wurden, lässt sich nicht ohne weiteres feststellen. Einfacher ist in diesem Fall die Anwendung von Verschlüsselungsverfahren, um solche Angriffe wirkungslos zu machen (vgl. Kapitel 2.6). Das Risiko für Angriffe auf die Datenintegrität kann durch eine Minimierung von Zugriffsrechten vermindert werden. Zur Erkennung solcher Angriffe existieren entsprechende Einbruchserkennungs-Systeme (*Intrusion Detection System (IDS)*), die innerhalb eines IT-Systems installiert werden können.
Zu erwähnen bleibt noch, dass es kein absolut sicheres System gibt. Jedoch kann das Risiko eines Schadensfalles durch eine erfolgreiche Sicherheitsrichtlinie erheblich minimiert werden. Die Umsetzung von IT-Sicherheit kostet Geld und um IT-Systeme abzusichern, können erhebliche Geldbeträge anfallen. Daher ist es notwendig, dass Aufwand und Nutzen von Sicherheitsvorkehrungen in einem angemessenen Verhältnis stehen.

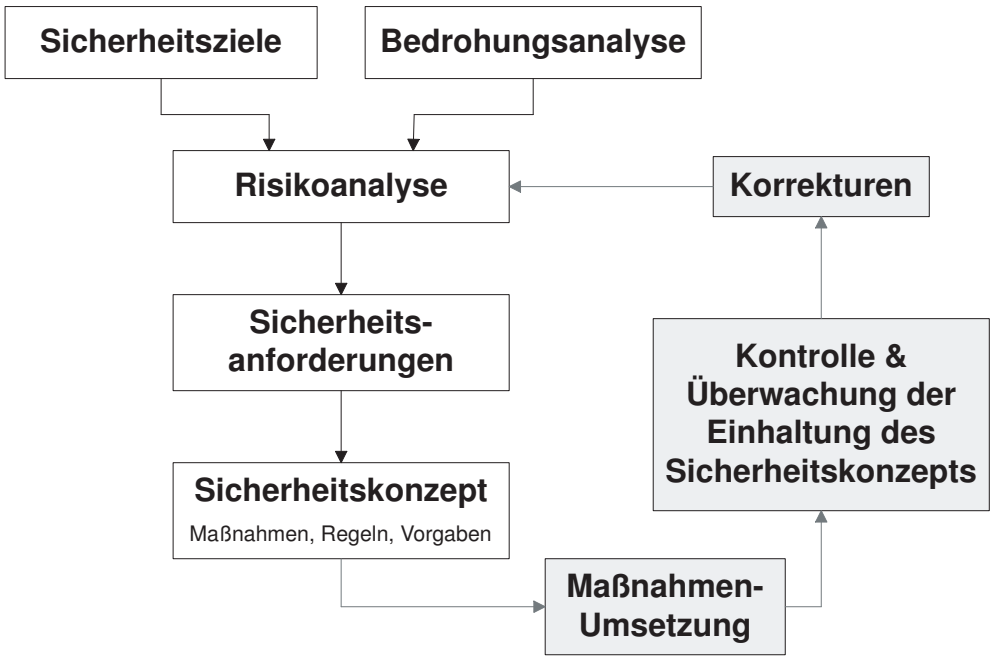

Abbildung 5.2: IT-Sicherheitsprozess

Um die IT-Sicherheit eines Systems auf längere Sicht zu gewährleisten, ist es notwendig, die Einhaltung des Sicherheitskonzepts immer wieder zu überprüfen und gegebenenfalls Korrekturen und Anpassungen vorzunehmen, die den Weiterentwicklungen der Technologien Rechnung tragen. IT-Sicherheit ist ein zyklischer Prozess (vgl. Abbildung 5.2), der seinen erreichten Sicherheitsstandard schnell wieder verliert, wenn er nicht konsequent weitergeführt wird.

5.1.2 Bedrohungsbaum

Für die Erfassung der Bedrohungen der SicAri-Plattform soll die Technik der Bedrohungs- bzw. Angriffsbäume (*Attack Tree*) eingesetzt werden. Die Attack Trees wurden erstmals im Jahr 1999

in Zusammenhang mit der Erfassung von Bedrohungen auf Computernetzwerke in einem Artikel [5] von Bruce Schneier diskutiert. Das Erstellen von Bedrohungsbäumen ist sehr stark an die Analyse von Fehlern durch Fehlerbäume im Bereich der Softwarezuverlässigkeit angelehnt. Mit einem Bedrohungsbaum können systematisch die Bedrohungen eines Systems erfasst werden. Zunächst wird die Wurzel des Baums definiert. Diese identifiziert eine mögliche Bedrohung des Systems und damit ein Angriffsziel. Auf der nächsten Ebene werden nun verschiedene Zwischenziele definiert, die zur Erreichung des Angriffsziels beitragen. Ob alle Zwischenziele erfüllt werden müssen oder ob jeweils schon ein Ziel ausreichend ist, um das höher liegende Ziel zu erfüllen, werden UND- bzw. ODER-Knoten verwendet. Ein UND-Knoten ist an der zusätzlichen Beschriftung der zugehörigen Kanten zu erkennen. ODER-Knoten werden im Bedrohungsbaum nicht besonders hervorgehoben.
Auf den weiteren Ebenen werden die einzelnen Ziele, bis hin zu den Blättern des Angriffsbaums, definiert. Die Blätter beschreiben jeweils einen einzelnen Angriffsschritt zum Erreichen eines Zwischenziels. Die Pfade von einem Blatt zur Wurzel des Baums beschreiben die unterschiedlichen Wege zum Erreichen des in der Wurzel definierten Angriffsziels. Die Abbildung 5.3 stellt den allgemeinen Aufbau eines Bedrohungsbaums dar. Zur Erreichung des Unterziels 1 in diesem Baum, reicht entweder die Aktion 1 oder 4 alleine bzw. eine Kombination der Aktionen 2 und 3 aus.
Zur besseren Darstellung wurde der Syntax eines Bedrohungsbaums um einige Elemente erweitert. So ist es möglich, dass ein Knoten des Baums mehrere Elternknoten besitzt. Außerdem werden die Knoten, die weitere Unterknoten oder Blätter enthalten, durch grau-hinterlegte Felder dargestellt.

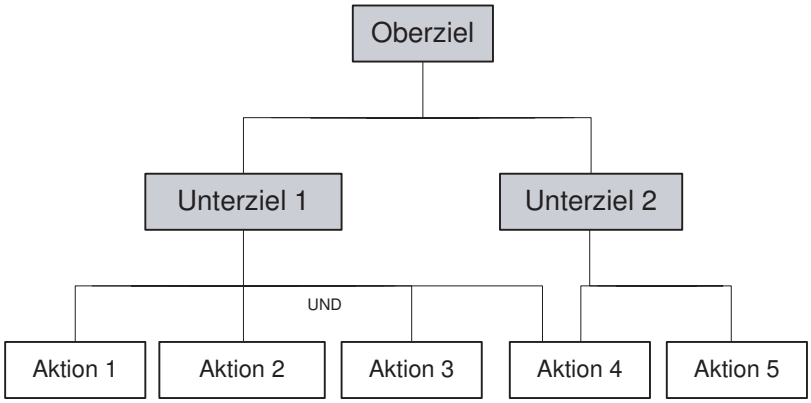

Abbildung 5.3: Grundlegender Aufbau eines Bedrohungsbaums

5.1.3 Kategorisierung von Bedrohungen

Zur Kategorisierung von Bedrohungen kann das so genannte STRIDE-Modell verwendet werden. Das von Microsoft (vgl. [65]) geprägte Akronym steht für die sechs Bedrohungskategorien:

- ***S**poofing*: Unter *Spoofing* oder auch *Masquerade* ist das Nachahmen bzw. Maskieren einer Identität zu verstehen. Ein Spoofing-Angriff zielt darauf ab, einem System eine falsche Identität vorzuspiegeln und so im Namen dieser Identität Aktionen durchzuführen. Spoofing-Angriffe werden in zwei Teilbereiche unterschieden. Einerseits kann ein

Angreifer versuchen, sich als ein regulärer Benutzer eines Systems auszugeben. Andererseits kann er die Identität eines Dienstes maskieren. Solch ein Angriff wäre beispielsweise das Maskieren der UDDI-Registry innerhalb einer SicAri-Infrastruktur. Spoofing-Angriffe sind damit Angriffe auf die Authentizität von Nutzern oder Systemen.

Ein Spoofing-Angriff kann auch ein so genannter *Man-in-the-Middle*-Angriff sein, der im nächsten Kapitel vorgestellt wird.

- ***T**ampering* oder *Falsification* bezeichnet einen Angriff, der auf die Verfälschung von Daten abzielt, das heißt ein Angreifer versucht unbefugt Änderungen an Daten vorzunehmen. Er zerstört damit die Integrität. Dies können Daten sein, die in einer Datenbank gespeichert sind oder über einen Kommunikationskanal ausgetauscht werden. Eine Verfälschung kann das Verändern, Löschen oder Hinzufügen von Daten bedeuten.

- ***R**epudiability*: Bei einem *Repudiability*-Angriff wird die Nicht-Abstreitbarkeit in Frage gestellt. Eine Aktion soll immer eindeutig einem Benutzer zugeordnet werden können, so dass dieser die Aktion nicht abstreiten kann. Dies ist damit ein Angriff auf die Verbindlichkeit.

- ***I**nformation disclosure* bzw. *Eavesdropping* bezeichnet die unerlaubte Informationsenthüllung bzw. das unerlaubte Abhören von Informationen. Ein Angriff dieser Kategorie ist ein Angriff auf die Vertraulichkeit.

- ***D**enial of service (DoS)*: Ein DoS-Angriff zielt darauf ab, die Verfügbarkeit eines Dienstes zu beeinträchtigen und führt, wenn er erfolgreich ist, zu einer Dienstverweigerung. DoS-Attacken werden meist durchgeführt, indem große Mengen von Nachrichten an einen Dienst gesendet werden

- **E**levation of privilege: Hierunter fallen Angriffe, die darauf abzielen, höhere Berechtigungen zu erlangen. Ein Angriff dieser Kategorie tritt beispielsweise ein, wenn ein Benutzer versucht Anwendungen oder Daten eines Systems aufzurufen, für die er im Normalfall keine Berechtigungen besitzt.

Die Bedrohungen lassen sich auch in die in [21] vorgeschlagenen Kategorien einordnen:

1. Bedrohungen durch externe Angriffe:
   Als externe Angriffe werden die Angriffe bezeichnet, die ohne die Hilfe des angegriffenen Systems durchgeführt werden. Hierunter fallen beispielsweise Zerstörung, Diebstahl oder das Beobachten einer Passwort-Eingabe.

2. Bedrohungen der Datenintegrität und Informationsvertraulichkeit:
   Alle Angriffe dieser Kategorie versuchen interne Kontrollen des Systems zu umgehen, um die Datenintegrität und Informationsvertraulichkeit zu verletzen.

3. Bedrohungen der Verfügbarkeit:
   Entspricht der Kategorie „DoS“ des STRIDE-Modells.

4. Abstreiten durchgeführter Aktionen:
   Entspricht der Kategorie „Repudiability“ des STRIDE-Modells.

5. Missbrauch erteilter Berechtigungen:
   Entspricht der Kategorie „Elevation of privilege“ des STRIDE-Modells.

Diese Einteilung ist sehr ähnlich wie das STRIDE-Modell aufgebaut. Die 2. Kategorie wird im STRIDE-Modell jedoch in zwei unterschiedliche Kategorien aufgeteilt. Im Folgenden werden die Bedrohungen anhand des STRIDE-Modells eingeteilt.

5.1.4 Bedrohungen

In diesem Kapitel werden gängige Angriffsmethoden bzw. Bedrohungen kurz erläutert:

- *Dictionary-Angriff*: Ein *Dictionary*-Angriff bezeichnet einen Angriff, der versucht, das Passwort eines Benutzers zu erraten. Hierbei wird automatisiert anhand eines Wörterbuchs eine Reihe von möglichen Passwörtern solange ausprobiert, bis das richtige Passwort gefunden ist. Diese Angriffe können durch einen verlangsamten Login-Prozess erschwert werden oder durch den Einsatz von so genannten *Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHAs)* komplett unterbunden werden. Ein CAPTCHA wird eingesetzt, um einen Benutzer von einem Computerprogramm unterscheiden zu können. Beim Login-Prozess wird dem Benutzers beispielsweise eine kleine Grafik angezeigt, die eine Folge von verschleierten Zeichen enthält. Diese Zeichen müssen beim Login eingegeben werden. Ein Benutzer kann normalerweise sehr einfach die angegebene Zeichenkette erkennen, ein Computerprogramm in der Regel nicht.

- *Replay-Angriff*: Eine *Replay*-Angriff zielt darauf ab, durch Wiedereinspielung von abgehörten Nachrichten die Authentizität des Datenursprungs zu verfälschen. Mit einer Replay Attack kann ein Angreifer versuchen, sich als ein regulärer Benutzer eines Systems auszugeben oder die zwischen zwei Instanzen ausgetauschten Informationen zu verfälschen.

- *Buffer Overflow-Angriff*: Eine *Buffer Overflow*-Angriff nutzt Schwachstellen von Systemen aus, die durch nachlässige Programmierung entstehen. Prüft ein Programm nicht die Eingabegröße eines Eingabefelds bevor es die Daten verarbeitet, kann durch gezielte Überschreitung dieser Eingabegröße ein Programmfehler erzeugt werden. Durch diesen kann die Verfügbarkeit des Systems beeinträchtigt werden oder sogar weiterer schädlicher Code eingeschleust werden.

- *Man-In-the-Middle (MITM)-Angriff*: Dieser Angriff ist eine besondere Form des Spoofing. Ein Angreifer setzt sich dabei zwischen zwei Kommunikationspartner wie Client und Server. Er gibt sich gegenüber dem Client als Server aus und gegenüber dem Server als Client. Die gesamte Kommunikation wird über den PC des Angreifers geleitet, der diese so abhören oder verfälschen kann.

- *Spoofing-Angriff*: Was ein Spoofing-Angriff ist, wurde bereits im vorherigen Kapitel erläutert. Spoofing-Angriffe können auf verschiedenen Ebenen erfolgen:

  - *IP-Spoofing*: Beim *IP-Spoofing* fälscht ein Angreifer die Quell- oder Ziel-Adresse eines IP-Datenpakets. Dadurch kann ein Angreifer die von ihm abgesendeten Pakete als Pakete ausgeben, die von einem als vertrauenswürdig angesehenen PC stammen. Er täuscht damit die Identität eines PCs vor.
  - *ARP-Spoofing*: Das *Address Resolution Protocol (ARP)-Spoofing* versucht, durch Manipulation des ARP-Caches einzelner Switches und Rechner in einem Netz, den PC eines Angreifers als regulären PC des Netzes auszugeben.
  - *DNS-Spoofing*: Beim *DNS-Spoofing* manipuliert ein Angreifer die Zuordnung von DNS-Namen zu IP-Adressen oder umgekehrt. So kann er einen PC maskieren.

- *SQL-Injection*: Bei diesem Angriff versucht ein Angreifer, durch das Einschleusen von eigenen Datenbankbefehlen, an Informationen zu gelangen oder diese zu verändern. Ein *SQL-Injection*-Angriff kann meistens dann erfolgreich gestartet werden, wenn nachlässig programmiert wurde. Werden beispielsweise die Felder eines Such-Formulars nicht ausreichend überprüft, kann ein Angreifer durch Eingabe von SQL-Befehlen direkt Manipulationen an der dahinterliegenden Datenbank vornehmen.

5.2 Durchführung

5.2.1 Identifikation der möglichen Bedrohungen

Die Bedrohunganalyse, die im Rahmen dieser Arbeit erstellt wird, umfasst nur eine Untersuchung der Benutzerauthentifikation der SicAri-Plattform. Eine detaillierte Analyse aller möglichen Bedrohungen der gesamten SicAri-Plattform bzw. einer kompletten SicAri-Infrastruktur ist nicht vorgesehen. Da dies sehr umfangreich ist, sollte dies in einer eigene Arbeit vorgenommen werden.
Die Bedrohunganalyse umfasst hier die Untersuchung der Authentifikations- und Autorisierungsmechanismen einer lokalen Plattform und diese in Kombination mit dem globalen Authentifikationsmanager. Dabei sind eine Reihe von Komponenten beteiligt, die, falls sie kompromittiert werden, die Authentifikation beeinträchtigen bzw. es möglich machen diese zu umgehen:

1. Externe Komponenten
   (a) Identitätsdatenbank
      - Globale „LDAP-Datenbank“
      - Dateibasierte Datenbank einer lokalen Plattform
   (b) UDDI-Registry
2. Interne Komponenten einer Plattform
   (a) Shell
   (b) Environment
   (c) SicAri-SecurityManager
3. Interne Services einer Plattform
   (a) Identitätsmanager
   (b) Schlüsselmanager
   (c) JAAS
   (d) lokaler Authentifikationsservice
   (e) Authentifikationsmanager
   (f) `AuthenticationManagerHandler`
   (g) `SecurityTokenHandler`
   (h) Policy-Service
4. Authentifikationsmethoden einer Plattform
   (a) Benutzername/Passwort

   (b) Soft Token (+ Passwort)
   (c) SmartCard (+ PIN)
   (d) Security Token

5. Weitere Angriffspunkte
   (a) Zusätzliche SicAri-Plattform/en
   (b) Web Service-spezifische Bedrohungen

Im Folgenden werden diese Angriffspunkte nun auf die damit verbundenen Bedrohungen analysiert. Die einzelnen Bedrohungen werden in Form von Listen aufgeführt. Können durch einen erfolgreichen Angriff weitere neue Angriffe durchgeführt werden, so werden diese als Unterpunkte gelistet. Für verschiedene Angriffspunkte werden außerdem Bedrohungsbäume erstellt. Die Bedrohungen werden, soweit dies sinnvoll ist, anhand des STRIDE-Modells eingeteilt. Ggf. werden auch Bedrohungen aufgeführt, die sich nicht in das STRIDE-Modell einteilen lassen.
Im anschließenden Kapitel wird aufgezeigt, wie die identifizierten Bedrohungen abgewehrt werden können.

5.2.1.1 Externe Komponenten

Identitätsdatenbank: Die für die Authentifikation benötigten Informationen, wie das öffentliche Zertifikat bzw. der Passwort-Hash-Wert eines Benutzers, sind zentral in einer über das *Lightweight Directory Access Protocol (LDAP)* angebundenen Datenbank gespeichert bzw. für eine Plattform, die keine Verbindung zu dieser Datenbank besitzt, in einer lokalen Datenbank in Form einer Datei abgelegt. Ein Angreifer kann durch gezielte Manipulation dieser Identitätsdatenbanken die Authentifikation von vorhandenen Benutzern verhindern oder durch Hinzufügen von Einträgen neuen Benutzern den Zugriff auf die Plattform ermöglichen. Durch Auslesen des Passwort-Hash-Werts aus der Datenbank und zusätzliche Manipulation der entsprechenden Login-Module, kann sich ein Angreifer unerlaubt an der Plattform anmelden und im Namen dieses Benutzers agieren. Die Datenbanken müssen daher vor unberechtigten Zugriffen besonders geschützt werden.
Des Weiteren kann eine Maskierung der „LDAP-Datenbank“ durch Techniken wie IP-, ARP-, DNS-Spoofing und Man-in-the-Middle-Attacken erfolgen. Außerdem kann durch entsprechende DoS-Attacken die Verfügbarkeit beeinträchtigt werden. Es ergeben sich die folgenden Bedrohungen.

- **T.id-db.spoofing**: Maskieren oder Ersetzen der Identitätsdatenbank
- **T.id-db.data-tampering**: Verändern der in der Identitätsdatenbank gespeicherten Informationen
- **T.id-db.data-eavesdropping**: Einsehen der in der Identitätsdatenbank gespeicherten Informationen
  - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
- **T.id-db.dos**: Stören der Verfügbarkeit oder Deaktivieren der Identitätsdatenbank

UDDI-Registry: Ein weiterer Anriffspunkt betrifft die UDDI-Registry von SicAri. Sie ist ein zentraler Punkt über den die verschiedenen Plattformen nach Diensten auf anderen Plattformen suchen. Gelingt es einem Angreifer gezielt eine Manipulation der Registry vorzunehmen, so kann er die gesamte SicAri-Infrastruktur beeinflussen. Beispielsweise wird über die Registry die WSDL-Beschreibung des Authentifikationsmanagers abgerufen. Ein Angreifer kann, durch die in der WSDL-Beschreibung enthaltenen Informationen, detaillierte Kenntnisse über den Aufbau dieses Dienstes erlangen und so gezielt Angriffe vorbereiten. Es kann daher sinnvoll sein, den Zugriff auf eine UDDI-Registry nur bestimmten Instanzen zu erlauben.
Außerdem kann der Angreifer durch Manipulation der Beschreibung die Verfügbarkeit eines Dienstes beeinflussen oder einen Dienst durch eine eigene Version ersetzen, in dem er die Beschreibung des Standorts eines Dienstes manipuliert.
Durch eine DoS-Attacke kann außerdem die Verfügbarkeit der UDDI-Registry gestört werden.

- **T.uddi.spoofing**: Maskieren oder Ersetzen der UDDI-Registry
- **T.uddi.data-tampering**: Verändern der Einträge der UDDI-Registry
- **T.uddi.data-eavesdropping**: Einsehen der Einträge der UDDI-Registry
- **T.uddi.dos**: Stören der Verfügbarkeit oder Deaktivieren der UDDI-Registry

In Abbildung 5.4 sind diese Bedrohungen nochmals in Form eines Bedrohungsbaums dargestellt.

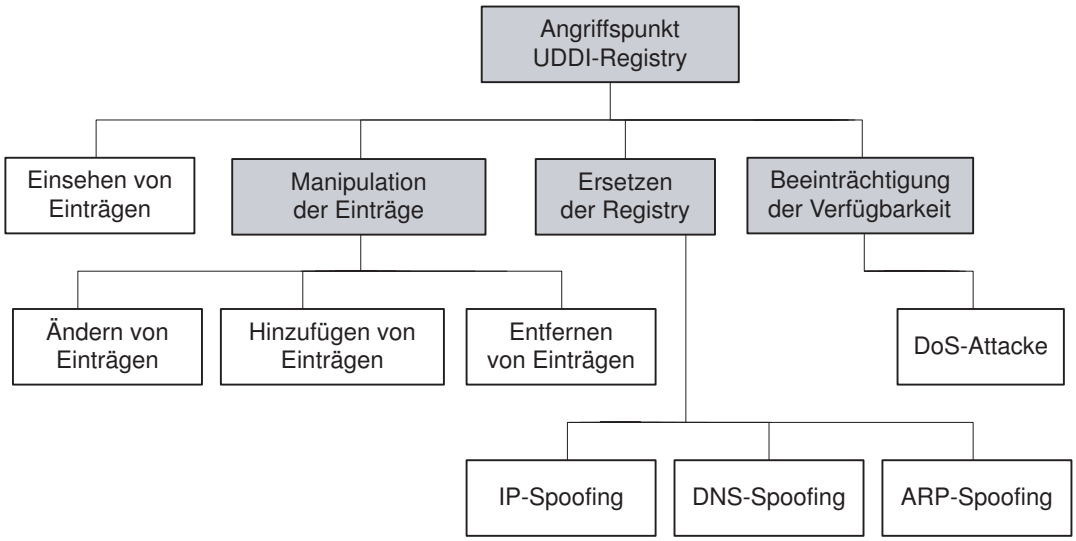

Abbildung 5.4: Bedrohungsbaum: UDDI-Registry

5.2.1.2 Interne Komponenten einer Plattform

Die Plattform besitzt eine Reihe von internen Angriffspunkten:

Shell: Gelingt es einem Angreifer die Shell einer Plattform durch eine eigene Version zu ersetzen, so kann er verschiedene Angriffe vornehmen. Zum einen kann er die Eingaben eines Benutzers mitschneiden und so in den Besitz von Passwörtern und anderen sicherheitsensitiven Informationen gelangen. Zum anderen kann er auch Befehle einschleusen und so die Aktionen des Benutzers verfälschen oder sogar Aktionen im Namen dieses Benutzers ausführen. Das führt dazu, dass nicht mehr eindeutig festgestellt werden kann, ob eine Aktion durch einen Benutzer oder einen Angreifer ausgeführt werden kann. Die Abstreitbarkeit wird in diesem Fall beeinflusst.

- **T.shell.spoofing**: Maskieren oder Ersetzen der Shell
  - **T.privilege-elevation**: Ausführen von Aktionen in Namen des angemeldeten Benutzers
  - **T.data-eavesdropping**: Ausspähen von Informationen
    - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
  - **T.shell.dos**: Stören der Verfügbarkeit oder Deaktivieren der Shell

Environment: Das Environment kann einen weiteren Angriffspunkt darstellen. Durch gezielte Manipulation kann ein Angreifer die Dienste, die einem Pfad zugeordnet sind, austauschen und die Zugriffsrechte verändern.

- **T.env.spoofing**: Maskieren oder Ersetzen des Environment
  - **T.env.data-tampering**: Ändern der Pfade des Environment
    - **T.env.privilege-elevation**: Ändern der Zugriffsrechte der einzelnen Pfade
- **T.env.dos**: Stören der Verfügbarkeit oder Deaktivieren des Environment

`SecurityManager`: Der SicAri-`SecurityManager` ist eine wichtige Komponente für die Durchsetzung der Sicherheitsrichtlinien der Plattform, da über den `SecurityManager` der Policy-Service konsultiert wird. Durch eine Veränderung des `SecurityManager` kann daher das Policy-Framework komplett deaktiviert werden. Der `SecurityManager` ist auch für die Überprüfung der Vertrauenswürdigkeit einer Klasse zuständig. Klassen, die nicht explizit als vertrauenswürdig über eine entsprechende Konfigurationsdatei festgelegt sind, verweigert der `SecurityManager` die Nutzung bzw. den Aufruf. Durch Manipulation der zugehörigen Konfigurationsdatei oder des `SecurityManager` selbst können daher auch Klassen ausgeführt werden, die nicht vertrauenswürdig sind.

- **T.sec-m.spoofing**: Maskieren oder Ersetzen des `SecurityManager`
  - **T.sec-m.privilege-elevation**: Ändern der Zugriffsrechte durch Deaktivierung der Sicherheitsrichtlinien
  - **T.sec-m.untrusted-code**: Einspielen von nicht-vertrauenswürdigen Klassen

5.2.1.3 Interne Services einer Plattform

Neben den internen Komponenten einer Plattform existieren verschiedene Dienste, die bei der Authentifikation von Benutzern beteiligt sind und ggf. Angriffsziele darstellen können:

Identitätsmanager: Der Identitätsmanager ist die Schnittstelle zu den bereits genannten Identitätsdatenbanken. Eine Manipulation dieses Dienstes kann daher die gleichen Auswirkungen wie eine Manipulation der Datenbanken selbst haben. Neben dem Maskieren des Identitätsmanagers kann ein Angreifer durch eine DoS-Attacke die Verfügbarkeit dieses Dienstes stören. Außerdem kann er die Daten, die über den Identitätsmanager aus der Datenbank an eine anfragende Instanz übergeben werden, manipulieren und einsehen.

- **T.id-m.spoofing**: Maskieren oder Ersetzen des Identitätsmanagers
  - **T.id-db.data-tampering**: Verändern der in der Identitätsdatenbank gespeicherten Informationen
  - **T.id-db.data-eavesdropping**: Einsehen der in der Identitätsdatenbank gespeicherten Informationen
- **T.id-m.dos:** Stören der Verfügbarkeit oder Deaktivieren des Identitätsmanagers

Für die Maskierung dieses Dienstes sowie die im Folgenden beschriebenen Dienste ergibt sich der in Abbildung 5.5 dargestellte Bedrohungsbaum.

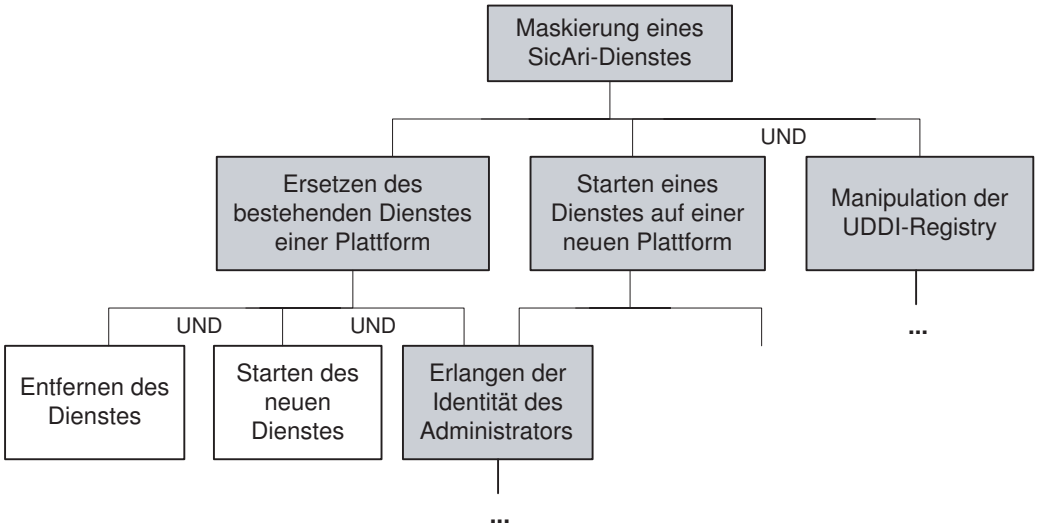

Abbildung 5.5: Bedrohungsbaum: SicAri-Services

Schlüsselmanager: Der Schlüsselmanager verwaltet die öffentlichen und privaten Schlüssel der Benutzer und ist damit ein potenzieller Angriffspunkt. Gelingt es einem Angreifer über den Schlüsselmanager auf die privaten Schlüssel eines Benutzers zuzugreifen, kann er diese missbrauchen. Auch die Veränderung oder das Löschen von Schlüsseln kann ein Angriffsziel darstellen. So kann ein Angreifer die Funktionen, für die ein Benutzer einen entsprechenden Schlüssel benötigt, einschränken.

- **T.key-m.spoofing**: Maskieren oder Ersetzen des Schlüsselsmanagers
  - **T.key.data-tampering**: Verändern von Schlüsseln
  - **T.key.data-eavesdropping**: Einsehen von Schlüsseln

        - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen

- **T.key-m.dos**: Stören der Verfügbarkeit oder Deaktivieren des Schlüsselmanagers

JAAS: Das JAAS-Framework ist ein weiterer Angriffspunkt. Durch Manipulation der zugehörigen Konfigurationsdatei kann ein Angreifer eigene Login-Module einschleusen oder die Parameter für vorhanden Login-Module verändern. Prinzipiell kann ein Angreifer durch ein manipuliertes Login-Modul die Eingaben eines Benutzers abhören und so in den Besitz von Passwörtern und privaten Zertifikaten gelangen.

- **T.jaas.spoofing**: Maskieren oder Ersetzen der Login-Module des JAAS
    - **T.jaas.data-eavesdropping**: Ausspähen von Anmeldeinformationen
        - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
- **T.jaas.data-tampering:** Verändern der Konfigurationsdatei des JAAS
- **T.jaas.dos**: Stören der Verfügbarkeit oder Deaktivieren des JAAS

Authentifikationsservice & Authentifikationsmanager: Genauso wie die anderen Dienste der Plattform, können auch der Authentifikationsservice und der Authentifikationsmanager angegriffen werden. Beide Dienste können durch eine eigene Version ersetzt oder manipuliert werden sowie durch entsprechend viele Anfragen einer DoS-Attacke unterzogen werden. Kann ein Angreifer einen dieser Dienste ersetzen, kann er dadurch die Authentifikation von Benutzern verhindern, Authentifikationsdaten erlangen oder neuen Benutzern den Zugriff gewähren.

- **T.auth-service.spoofing**: Maskieren oder Ersetzen des Authentifikationsservice
    - **T.auth-service.data-eavesdropping**: Ausspähen von Anmeldeinformationen
        - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
- **T.auth-service.dos**: Stören der Verfügbarkeit oder Deaktivieren des Authentifikationsservice

Für die Web Service-Schnittstelle des Authentifikationsmanagers wurde ein Bedrohungsbaum erstellt, der in Abbildung 5.6 dargestellt ist. Neben der Maskierung des Dienstes (vgl. Bedrohungsbaum in Abbildung 5.5) kann der Authentifikationsmanager auch einer DoS-Attacke unterzogen werden. Außerdem kann zum Erlangen einer Identität die Kommunikation abgehört werden und danach eine Replay-Attacke gestartet werden oder ein Man-in-the-Middle-Angriff durchgeführt werden.

- **T.auth-manager.spoofing**: Maskieren oder Ersetzen des Authentifikationsmanagers
    - **T.auth-manager.data-eavesdropping.1**: Ausspähen von Anmeldeinformationen
        - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
- **T.auth-manager.data-eavesdropping.2**: Ausspähen von Anmeldeinformationen durch Abhören des Kommunikation
    - **T.auth-manager.replay:** Wiedereinspielen von abgehörten Anmeldeinformationen

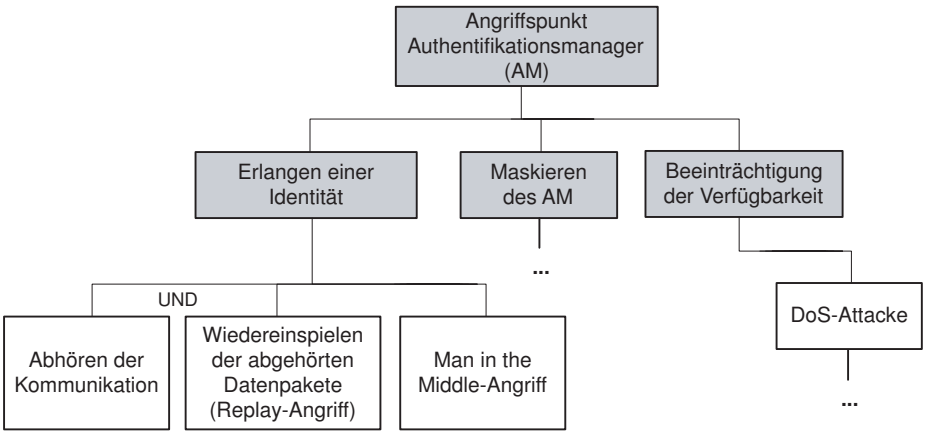

Abbildung 5.6: Bedrohungsbaum: Authentifikationsmanager

    - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen

- **T.auth-manager.mitm:**
  - **T.auth-manager.data-eavesdropping**: Ausspähen von Anmeldeinformationen
    - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
- **T.auth-manager.dos**: Stören der Verfügbarkeit oder Deaktivieren des Authentifikationsmanagers

Policy-Service: Wie bereits bei den Bedrohungen des `SecurityManager` beschrieben, können die Sicherheitsrichtlinien der Plattform umgegangen werden, in dem der Policy-Service einfach nicht mehr konsultiert wird. Der Policy-Service kann jedoch auch selbst manipuliert werden. Ein Angreifer kann den Policy-Service durch eine eigene Version ersetzen oder den bestehenden manipulieren. Dafür kann er die Konfigurationsdateien, in denen die Benutzerrollen und zugehörigen Berechtigungen definiert sind, verändern. Auch eine DoS-Attacke auf den Policy-Service ist problematisch, da dadurch die Verfügbarkeit des Dienstes beeinflusst wird. Dies kann dazu führen, dass die Entscheidung, ob eine Benutzeraktion erlaubt oder nicht erlaubt ist, erheblich verzögert wird.

- **T.policy.spoofing**: Maskieren oder Ersetzen des Policy-Service
  T.policy.data-tampering. Verändern der Sicherheitsrichtlinien-Definition
  - **T.data-eavesdropping**: Ausspähen von Informationen
  - **T.privilege-elevation**: Ausführen von Aktionen, die durch den Policy-Service normalerweise unterbunden werden
- **T.policy.dos**: Stören der Verfügbarkeit oder Deaktivieren des Policy-Service

Axis-Handler STH & AMH: Die Axis-Handler STH und AMH stellen weitere Angriffspunkte dar. Durch Manipulation, Entfernen oder Ersetzen dieser Handler kann die Authentifikation über den Authentifikationsmanager beeinflusst und der SSO kompromittiert werden. Da beide Handler Security Tokens verarbeiten, kann ein Angreifer über die Handler unbefugt an

ein ST gelangen, das er dann für seine Zwecke missbrauchen kann. Der AMH besitzt zudem zwei öffentliche, statische Methoden, über die Security Tokens vom AMH abgerufen bzw. dort zwischen gespeichert werden können (siehe Kapitel 4.2.1.4). Dies stellt eine erhebliche Sicherheitslücke dar, ist jedoch unvermeidbar, um die Kommunikation mit den Authentifikationsmanger und dem Authentifikationsservice zu gewährleisten. Wie diese gesichert werden kann, wird in Kapitel 5.2.2.2 (vgl. D.2.18) erläutert.
Im Zusammenhang mit den Axis-Handler existiert eine weitere Angriffsmöglichkeit. Theoretisch kann ein Angreifer über die entsprechenden Shell-Kommandos weitere Handler in die verschiedenen Handlerketten einbinden. So kann er ggf. alle Nachrichten, die über das WS-Framework verarbeitet werden, einsehen, verändern oder sogar löschen. Hierbei ist auch die Position des Handler in der Kette zu beachten. Durchläuft eine Nachricht zunächst den STH oder AMH und nehmen diese Handler eine Verschlüsselung der Nachricht vor, kann der nächste Handler in der Kette die Nachricht natürlich nicht mehr ohne weiteres einsehen. Wird der Handler des Angreifers jedoch vor dem AMH oder STH platziert, kann der Handler die Nachrichten problemlos verarbeiten.

- **T.handler.spoofing**: Maskieren oder Ersetzen eines Handler
  - **T.data-tampering**: Verändern der SOAP-Nachrichten
  - **T.data-eavesdropping**: Einsehen der SOAP-Nachrichten
    - **T.identity.spoofing**: Anmelden mit den ausgespähten Informationen
- **T.amh.st-eavesdropping**: Abruf von Security Tokens über die Schnittstelle des AMH

5.2.1.4 Authentifikationsmethoden einer Plattform

Das Vortäuschen einer falschen Identität bzw. das Einnehmen einer anderen Identität kann durch Ausnutzung der verschiedenen Authentifikationsmethoden auf unterschiedliche Weise erfolgen:

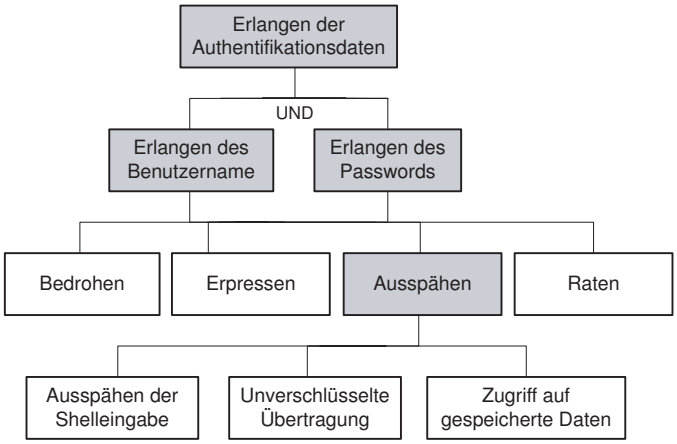

Abbildung 5.7: Bedrohungsbaum: Benutzername/Password-Authentifikation ($M1$)

M1: Für die Authentifikationsmethode $M1$ wurde der in Abbildung 5.7 dargestellte Bedrohungsbaum erstellt. Ein Angreifer benötigt für einen erfolgreichen Angriff den Namen und das

zugehörige Passwort eines Benutzers. An diese Daten kann er durch Bedrohen oder Erpressen des Benutzers oder durch Raten gelangen. Außerdem kann ein Angreifer versuchen, diese Daten auszuspähen. Dazu kann er versuchen, die Shell-Eingaben auszuspähen oder die Daten einzusehen, falls diese über einen unverschlüsselten Kanal übertragen werden. Falls die Daten in gespeicherter Form vorliegen, kann der Angreifer außerdem die Daten über diesen Weg einsehen. Für die Authentifikationsmethode $M1$ benötigt der Angreifer nicht unbedingt das Passwort des Benutzers, denn für die erfolgreiche Authentifikation anhand der in Protokollspezifikation $AP1$ definierten Schritte, reicht die Kenntnis über den Hash-Wert des Passworts aus.
Die Angriffe durch Erpressen, Bedrohen oder Inbesitznahme können durch technische Maßnahmen des Systems nicht verhindert werden. Diese Angriffe werden daher in der weiteren Betrachtung nicht detailliert.

- **T.pwd.eavesdropping**: Ausspähen des Passworts
- **T.pwd.dictionary**: Raten des Passworts

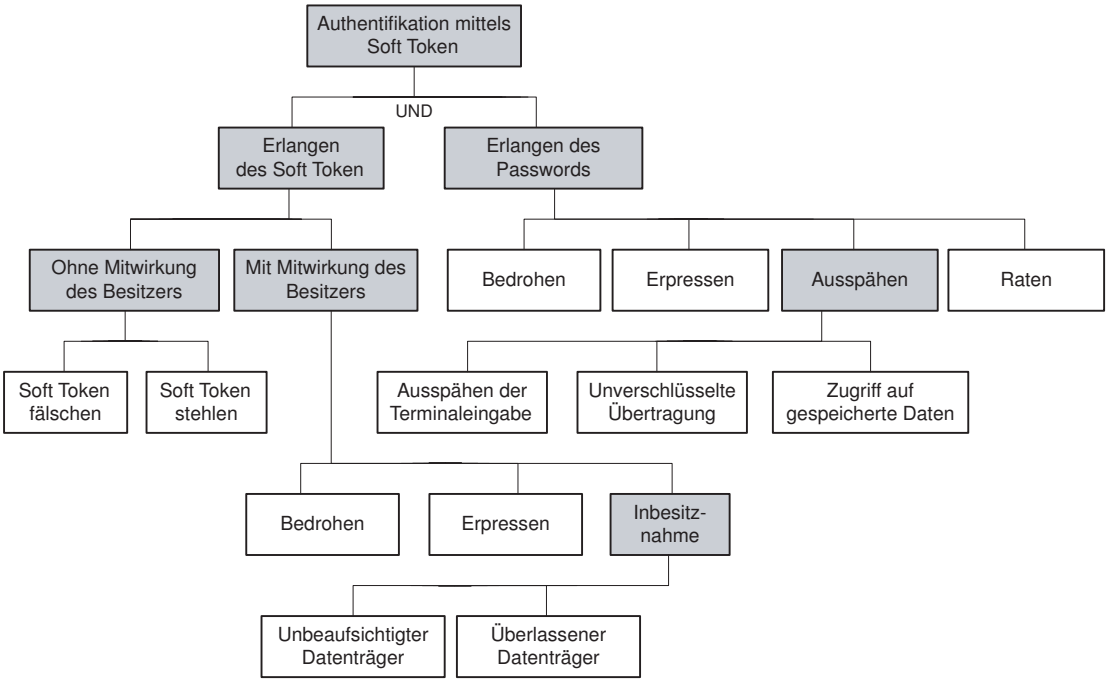

Abbildung 5.8: Bedrohungsbaum: Soft Token-Authentifikation ($M2$)

M2: Abbildung 5.8 zeigt den Bedrohungsbaum für die Authentifikationsmethode $M2$. Der Baum zeigt, dass für die erfolgreiche Authentifikation das zugehörige Passwort des Soft Token benötigt wird. Dies kann der Angreifer auf die gleiche Weise erlangen, wie bereits für die Methode $M1$ erläutert. An das Soft Token kann der Angreifer entweder ohne oder unter Mitwirkung des Besitzers gelangen. Im ersten Fall kann er versuchen, das Soft Token zu stehlen oder zu fälschen. Anderenfalls kann er versuchen, den Benutzer um die Herausgabe des Soft Token zu bedrohen oder zu erpressen. Des Weiteren kann der Angreifer auch durch einen unbeaufsichtigten oder überlassenen Datenträger in den Besitz des Soft Token gelangen. Ist der Angreifer in Besitz des Soft Token, kann er versuchen, an das korrekte Passwort durch Raten zu gelangen.

- **T.softtoken.eavesdropping**: Erlangen des Soft Token
- **T.softtoken.pwd.eavesdropping**: Ausspähen des Passworts
- **T.softtoken.pwd.dictionary**: Raten des Passworts
- **T.softtoken.tampering**: Fälschen eines Soft Token

M3: Der Bedrohungsbaum der Authentifikationsmethode $M3$ in Abbildung 5.9 unterscheidet sich kaum von dem der Authentifikationsmethode $M2$. Hier muss der Angreifer in den Besitz der Smart Card und der zugehörigen PIN gelangen.

- **T.smartcard.eavesdropping**: Erlangen der Smart Card
- **T.smartcard.pin.eavesdropping**: Ausspähen der PIN
- **T.smartcard.pin.dictionary**: Raten der PIN
- **T.smartcard.tampering**: Fälschen einer Smart Card

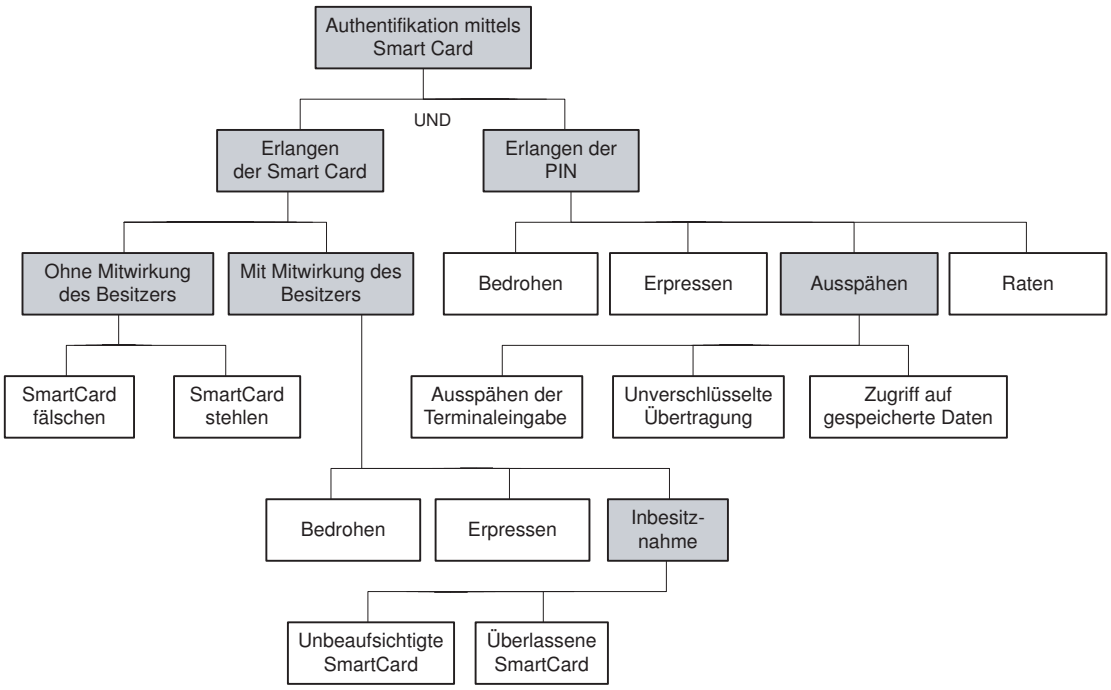

Abbildung 5.9: Bedrohungsbaum: Smart Card-Authentifikation ($M3$)

M4: Für die Authentifikationsmethode $M4$ wurde der in Abbildung 5.10 dargestellte Bedrohunsbaum erstellt. Um sich erfolgreich mit einem Security Token zu authentifizieren, muss der Angreifer, im Gegensatz zu den vorherigen Authentifikationsmethoden, nur in Besitz des Security Token gelangen. An dieses kann er gelangen, in dem er das Security Token fälscht oder versucht es zu stehlen. Dies ist über eine unterschlüsselte Übertragung oder durch den Zugriff auf ein gespeichertes ST möglich. Der Angreifer kann den Benutzer zur Herausgabe des ST

bewegen, indem er ihn bedroht oder erpresst. Neben einem gültigen ST kann der Angreifer auch versuchen, an ein bereits abgelaufenes ST zu gelangen. Für die erfolgreiche Authentifikation mit dem abgelaufenen ST muss der Angreifer zusätzlich die Systemzeit des Computers manipulieren, an dem er sich anmelden möchte.
Ein ST besitzt nur eine festgelegte Gültigkeit. Gelingt es einem Angreifer an ein gültiges ST zu gelangen, kann er sich damit an einer Plattform authentifizieren. Er kann dann die Dienste der Plattform solange nutzen, bis er sich wieder abmeldet, denn die Plattform prüft derzeit nach der Authentifikation nicht erneut die Gültigkeit des ST.

- **T.st.eavesdropping**: Ausspähen eines Security Token
  - **T.st.invalid**: Nutzung eines abgelaufenen ST zur Maskierung eines Benutzers
- **T.st.tampering**: Fälschen eines Security Token
- **T.st.runtime**: Überschreitung der ST-Laufzeit

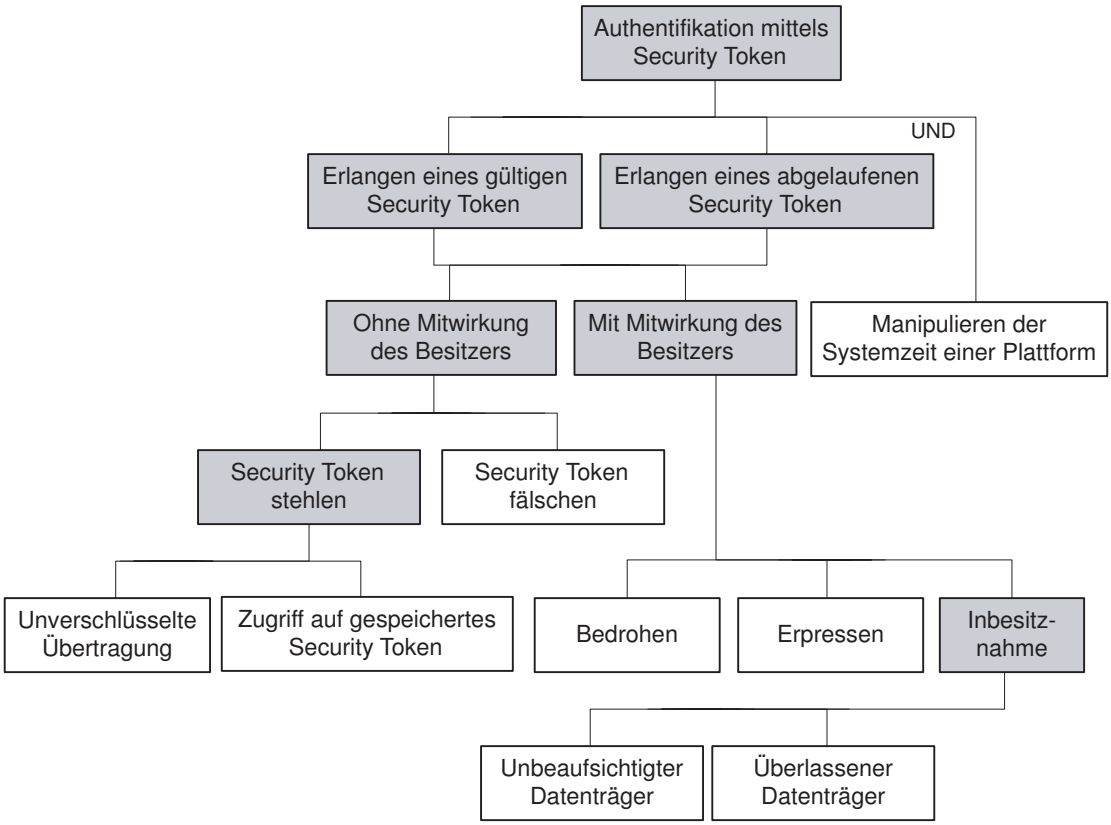

Abbildung 5.10: Bedrohungsbaum: Security Token-Authentifikation ($M4$)

Das Ausspähen der Authentifikationdaten kann auch durch das Abhören der Kommunikation über die Web-Service-Schnittstelle des Authentifikationsmanages erfolgen. Ein Angreifer kann so die Authentifikationsanfrage ausspähen und eine Replay-Attacke starten. Auch eine MITM-Angriff ist hier denkbar:

- **T.auth.replay**: Wiedereinspielen von ausgespähten Authentifikationsanfragen

- **T.auth.mitm**: Man-in-the-Middle-Angriff zum Maskieren einer Identität

Neben diesen Bedrohungen der einzelnen Authentifikationsmethoden ergeben sich, durch das Vortäuschen einer Identität, eine Reihe von weiteren abhängigen Bedrohungen:

- **T.identity.spoofing**
  - **T.repudiability**: Hat sich ein Angreifer maskiert, kann nicht mehr sichergestellt werden, ob eine Aktion im Namen des Angreifers oder des regulären Benutzers gestartet wurde.
  - **T.privilege-elevation**: Durch die Maskierung gelangt der Angreifer an Rechte, die er normalerweise nicht besitzt. Er erweitert damit seine Berechtigungen.
  - **T.data-tampering**: Hat sich ein Angreifer erfolgreich authentifiziert, kann er unberechtigt Veränderung an Daten vornehmen.
  - **T.data-eavesdropping**: Durch eine erfolgreiche Maskierung kann der Angreifer auch Informationen einsehen, die nicht für ihn bestimmt sind.

5.2.1.5 Weitere Angriffspunkte

Zusätzliche SicAri-Plattform/en Gelingt es einem Angreifer eine weitere SicAri-Plattform zu starten, und wird diese von den anderen Plattformen als vertrauenswürdig angesehen, kann der Angreifer verschiedene Dienste der Infrastruktur auf seiner Plattform starten und über die UDDI-Registry veröffentlichen. Er kann durch das Maskieren einer Plattform also auch verschiedene Dienste der Infrastruktur maskieren.

- **T.platform.spoofing**: Maskieren oder Ersetzen einer SicAri-Plattform

Web Service-spezifische Bedrohungen In der Literatur ([42, 105, 16]) lassen sich eine Reihe von Web Service-spezifischen Bedrohungen finden, die auch für die Web Services der SicAri-Plattform relevant sein können:

- **T.wsdl.data-eavesdropping** *WS-Interface Probing*: Diese Bedrohung wurde bereits im Zusammenhang mit der UDDI-Registry erwähnt. Durch eine WSDL-Beschreibung erhält ein Angreifer detaillierte Kenntnisse über die Schnittstelle des Web Service, den er angreifen möchte. Er kann damit gezieltere Angriffe vorbereiten.
- **T.ws.xml-dos** (*XML-DoS*): Dieser Angriff bezieht sich auf das Parsen von XML-Nachrichten. Werden sehr große XML-Nachrichten verarbeitet, kann das Parsen sehr zeit- und ressourcenintensiv sein. Dadurch kann ggf. der Parser überladen und ein DoS erreicht werden.
- **T.ws.malicious-content** (*Malicious Content*): XML-Dokumente können Parsing - oder Verarbeitungsanweisungen (XML-Schema Erweiterungen, XPath-, XQuery-, XSLT-Kommandos) enthalten, die ggf. bösartige Aktionen auf dem verarbeiteten System zur Folge haben.
- **T.ws.parameter-attacks** (*Parameter Attacks*): Hierunter sind alle Bedrohungen zu verstehen, die sich durch die Parameter einer Funktion ergeben, die als Web Service bereitgestellt wird. Ein Angreifer kann versuchen, durch ausreichend große Eingabeparameter, die Speichergröße eines Datentyps zu überschreiten und so einen Buffer-Overflow-Angriff durchzuführen. Auch SQL-Injection-Angriffe sind hier denkbar.

- **T.ws.unauthorized-access** (*Unauthorized Access*): Hierunter fallen alle Bedrohungen durch nicht-autorisierte Zugriffe auf einen Web Service. Für die Nutzung von Web Services muss deshalb ggf. zuvor eine Authentifikation von Benutzern erfolgen.

- **T.ws.recursive-references** (*Rekursive Referenzen*): XML-Dokumente können so genannte *Entities* enthalten. Diese definieren ein Schlüsselwort, das beim Einlesen des XML-Dokuments mit dem definierten Entity-Wert ersetzt wird. Sie sind also eine Art Platzhalter bzw. Makro. Entities werden durch das Element `<!ENTITY ...>` definiert. Werden innerhalb eines XML-Dokuments rekursive Entities verwendet, kann das beim Einlesen zu einem DoS führen.

- **T.ws.external-references** (*Externe Referenzen*): XML-Entities können auch Referenzen auf externe Quellen enthalten, die beim Parsen abgerufen werden. Ein Angreifer kann versuchen, mittels solcher Referenzen auf lokale Dateien zu verweisen, um so ggf. über eine Fehlermeldung eine Kopie des Inhalts zu erhalten.

- **T.ws.protocol-threats** (*SOAP/XML Protokoll*): Da Web Services auf Basis verschiedener Transportprotokolle implementiert werden können, sind hier jeweils auch protokollspezifische Bedrohungen zu erwarten. Welche Bedrohungen dies im einzelnen sind, muss anhand des eingesetzten Protokolls untersucht werden.

- **T.ws.security-tampering** (*XML Security Tampering*): Der Einsatz von Techniken wie XMLdsig und XMLenc garantiert keine absolute Sicherheit. Durch die Verwendung von schwachen Schlüsseln oder unzureichend sicheren Algorithmen kann eine Signatur manipuliert werden oder eine Verschlüsselung gebrochen werden.

- **T.ws.end-to-end-security** (*Ende-zu-Ende-Sicherheit*): Da SOAP-Nachrichten ggf. über Zwischenknoten bis zu ihrem eigentlichen Bestimmungsort gesendet werden können, ist die Sicherung der Vertraulichkeit und Integrität mittels traditionellen Techniken wie SSL/TLS oder IPsec nicht möglich. Die Nachrichten liegen an den einzelnen Verarbeitungsknoten im Klartext vor und können ggf. von einem Angreifer eingesehen werden.

5.2.2 Abwehr der Bedrohungen

5.2.2.1 Externe Komponenten

Bedrohungen	Abwehr
T.id-db.spoofing	D.1.1
T.id-db.data-tampering	D.1.2
T.id-db.data-eavesdropping	D.1.2 & D.1.3
- T.identity.spoofing	D.1.4
T.id-db.dos	D.1.5
T.uddi.spoofing	D.1.1
T.uddi.data-tampering	D.1.2
T.uddi.data-eavesdropping	D.1.2 & D.1.3
T.uddi.dos	D.1.5

Tabelle 5.1: Zuordnung Bedrohungen und Abwehrmaßnahmen: Externe Komponenten

- **D.1.1**: Die Spoofing-Angriffe auf die Identitätsdatenbank bzw. UDDI-Registry können nur durch klassische Mechanismen, wie Firewalls bzw. Paketfilter, erkannt und abgewehrt werden. Für den Fall des ARP-Spoofing können intelligente Programme installiert werden, die die ARP-Pakete auf ihre Sinnhaftigkeit untersuchen.

- **D.1.2**: Der Zugriff auf die Datenbank bzw. UDDI-Registry muss durch Authentifikationsmechanismen abgesichert werden. Nur wenn diese überwunden werden können ist es möglich, die gespeicherten Informationen zu manipulieren bzw. einzusehen.

  Derzeit ist der Zugriff auf die „LDAP-Datenbank“ durch eine Benutzername/Passwort-basierende Authentifikation gesichert. Die Zugriffsdaten werden jedoch im Klartext in der Konfigurationsdatei `identity.ldap` abgespeichert. Dies stellt ein hohes Risiko dar. Außerdem ist die dateibasierte lokale Datenbank im Klartext abgespeichert. Hier ist eine verschlüsselte Abspeicherung einzusetzen, um die Manipulation dieser sensiblen Daten zu verhindern.

  Für die UDDI-Registry ist ebenfalls eine Benutzername/Passwort-basierende Authentifikation erforderlich. Der Zugriff auf die Registry ist für einen Angreifer deshalb nicht ohne Weiteres möglich.

- **D.1.3**: Kommuniziert eine Plattform mit der „LDAP-Datenbank“ bzw. UDDI-Registry, muss diese Kommunikation vor Verletzungen der Integrität und Vertraulichkeit durch entsprechende Maßnahmen, wie zum Beispiel die Nutzung des SSL/TLS-Protokolls, abgesichert werden. Die Plattform bietet sowohl die Möglichkeit der ungesicherten als auch der mittels SSL/TLS gesicherten Kommunikation mit der „LDAP-Datenbank“ bzw. UDDI-Registry. Die gesicherte Kommunikation ist einer unverschlüsselten Übertragung in jedem Fall vorzuziehen, da ansonsten der Benutzername und das Passwort abgehört werden können.

  Bei Verwendung einer lokalen dateibasierten Identitätsdatenbank muss diese durch Verschlüsselung vor unerlaubtem Einsehen geschützt werden.

- **D.1.4**: Voraussetzung für das Maskieren einer Identität ist das unerlaubte Einsehen der Identitätsdatenbank. Dies kann durch die Abwehrmechanismen D.1.2 und D.1.3 abgewehrt werden.

- **D.1.5**: DoS-Angriffe auf die UDDI-Registry bzw. die „LDAP-Datenbank“ können mit Mechanismen wie Proxies und Paketfiltern größtenteils erkannt und abgewehrt werden. DoS-Angriffe können jedoch auch durch fehlerhafte Implementierungen einzelner Systemkomponenten möglich werden. Die Registry bzw. die Datenbank müssen deshalb entsprechend fehlerfrei sein und ggf. auch eigene Schutzmechanismen besitzen.

5.2.2.2 Interne Komponenten und Services einer Plattform

Bedrohungen	Abwehr
T.shell.spoofing	D.2.1
- T.privilege-elevation	D.2.2
- T.data-eavesdropping	D.2.3
- T.identity.spoofing	D.2.4
- T.shell.dos	D.2.5

Bedrohungen	Abwehr
T.env.spoofing	D.2.1
- T.env.data-tampering	D.2.6
- T.env.privilege-elevation	D.2.2
T.env.dos	D.2.7
T.sec-m.spoofing	D.2.1
- T.sec-m.privilege-elevation	D.2.2
- T.sec-m.untrusted-code	D.2.8
T.id-m.spoofing	D.2.1
- T.id-db.data-tampering	D.2.9
- T.id-db.data-eavesdropping	D.2.3
- T.identity.spoofing	D.2.4
T.id-m.dos	D.2.10
T.key-m.spoofing	D.2.1
- T.key.data-tampering	D.2.11
- T.key.data-eavesdropping	D.2.3
- T.identity.spoofing	D.2.4
T.key-m.dos	D.2.10
T.jaas.spoofing	D.2.1
- T.jaas.data-eavesdropping	D.2.3
- T.identity.spoofing	D.2.4
T.jaas.data-tampering	D.2.12
T.jaas.dos	D.2.10
T.auth-service.spoofing	D.2.1
- T.auth-service.data-eavesdropping	D.2.3
- T.identity.spoofing	D.2.4
T.auth-service.dos	D.2.10
T.auth-manager.spoofing	D.2.1
- T.auth-manager.data-eavesdropping.1	D.2.3
- T.identity.spoofing	D.2.4
T.auth-manager.replay	D.2.13 & D.3.12
- T.identity.spoofing	D.2.4
T.auth-manager.mitm	D.2.14 & D.3.13
- T.identity.spoofing	D.2.4
T.auth-manager.dos	D.2.10 & D.2.15 & D.4.2
T.policy.spoofing	D.2.1
& T.policy.data-tampering	D.2.16
- T.data-eavesdropping	D.2.3
- T.privilege-elevation	D.2.2
T.policy.dos	D.2.10
T.handler.spoofing	D.2.1
- T.data-tampering	D.2.17
- T.data-eavesdropping	D.2.3
- T.identity.spoofing	D.2.4
T.amh.st-eavesdropping	D.2.18

Tabelle 5.2: Zuordnung Bedrohungen und Abwehrmaßnahmen: Interne Komp. & Services

- **D.2.1**: Um einen Spoofing-Angriff auf die internen Dienste einer Plattform erfolgreich durchführen zu können, muss ein Angreifer Administrationsrechte besitzen. Besitzt er diese nicht, werden alle Aktionen zum Neustarten oder Ersetzen von Diensten, durch den Policy-Service bzw. die definierten Sicherheitsrichtlinien, mit einer Fehlermeldung quittiert.

  Voraussetzung für diesen Angriff ist daher die erfolgreiche Authentifikation als Administrator. Im nächsten Kapitel wird gezeigt, welche Mechanismen die Plattform bietet, um dies zu verhindern.

- **D.2.2**: Die Erweiterung der Berechtigung ist nur dann möglich, wenn einem Angreifer die Maskierung der Shell, des `SecurityManager` oder des Policy-Service gelingt. Dieser Angriff kann daher durch die in D.2.1 beschriebenen Mechanismen abgewehrt werden.

- **D.2.3**: Für das unerlaubte Einsehen von Daten ist die erfolgreiche Maskierung eines Axis-Handler, des Policy-Service, des Identitäts- oder Schlüsselmanagers, des JAAS, des Authentifikationsservice, des Authentifikationsmanagers oder der Shell Voraussetzung. Dieser Angriff kann daher durch die in D.2.1 beschriebenen Mechanismen abgewehrt werden.

- **D.2.4**: Voraussetzung für das Maskieren einer Identität ist das Ausspähen oder Fälschen von Authentifikationsdaten. Theoretisch kann ein Angreifer durch das Maskieren verschiedener Dienste der Plattform an diese Daten gelangen. Um jedoch einen Dienst erfolgreich maskieren zu können, benötigt der Angreifer Administrationsrechte. Nur so kann er einen Dienst durch eine eigene Version ersetzen. Wie abgewehrt werden kann, dass der Angreifer in den Besitz von Administrationsrechten gelangt, wird im nächsten Kapitel erläutert.

- **D.2.5**: Da für jeden Benutzer eine eigene Shell gestartet wird, kann ein Angreifer in einem Angriffschritt jeweils nur die Verfügbarkeit einer einzigen Shell beeinträchtigen. Voraussetzung ist jedoch das Maskieren einer Shell. Dieser Angriff kann daher durch die in D.2.1 beschriebenen Mechanismen abgewehrt werden.

- **D.2.6**: Um die Pfade im Environment zu verfälschen, muss es einem Angreifer gelingen, das Environment zu maskieren. Dies kann durch die in D.2.1 beschriebenen Mechanismen abgewehrt werden.

- **D.2.7**: Um einen erfolgreichen DoS-Angriff auf das Environment vornehmen zu können, muss sich der Angreifer zunächst als Benutzer der Plattform angemeldet haben. Danach kann er versuchen, in einer Endloschleife Anfragen an das Environment zu stellen. Wie solche Angriffe abgewehrt werden können, wird hier nicht weiter detailliert, da sie nicht die Benutzerauthentifikation betreffen.

- **D.2.8**: Gelingt es einem Angreifer die Konfigurationsdatei des `SecurityManager` zu manipulieren, kann er Code einschleusen, der normalerweise nicht als vertrauenswürdig angesehen wird. Dadurch ergeben sich weitere Bedrohungen. Zur Abwehr sollte die Konfigurationsdatei in verschlüsselter Form gespeichert werden. Mit Hilfe einer digitalen Signatur kann außerdem eine evtl. vorliegende Manipulation erkannt werden.

- **D.2.9**: Damit ein Angreifer über den Identitätsmanager Manipulationen an der Identitätsdatenbank vornehmen kann, muss er diesen Dienst zunächst maskieren. Durch die in D.2.1 dargestellten Mechanismen kann dies verhindert werden.

- **D.2.10**: Um die Verfügbarkeit des Authentifikationsservice, Identitätsmanager, Schlüsselmanager, Authentifikationsmanager, Policy-Service oder des JAAS zu stören, muss ein Angreifer entsprechend viele gleichzeitige Anfragen an diese Dienste stellen. Durch das Proxy-Konzept (vgl. Kapitel 3.4.1.2) wird die Anfälligkeit dieser Dienste gegenüber DoS-Attacken reduziert. Der JAAS ist jedoch nicht durch einen Proxy geschützt. Eine DoS-Attacke kann hier nur durch eine verzögerte Anmeldung abgewehrt werden.

- **D.2.11**: Um über den Schlüsselmanager die einzelnen Schlüssel manipulieren zu können, muss der Angreifer wieder eine Maskierung dieses Dienstes vornehmen. Dieser Angriff kann daher durch die in D.2.1 beschriebenen Mechanismen abgewehrt werden.

- **D.2.12**: Die Konfigurationsdatei des JAAS-Framework ist hier der Hauptangriffspunkt. Zur Abwehr sollte diese verschlüsselt werden. Es besteht auch die Möglichkeit, für die Konfigurationsdatei eine Signatur zu berechnen. Beim Einlesen der Datei kann diese Signatur dann überprüft werden und so eine vorhandene Manipulation erkannt werden.

- **D.2.13**: Da für den Authentifikationsmanager eine Web Service-Schnittstelle existiert, kann ein Angreifer darüber die Kommunikation abhören und so versuchen, eine Replay-Attacke durchzuführen. Wie dies abgewehrt werden kann, wird in Kapitel 5.2.2.3 (vgl. D.3.12) dargestellt.

- **D.2.14**: Die Web Service-Schnittstelle bietet einem Angreifer auch die Möglichkeit, eine MITM-Attacke durchzuführen. Wie dies abgewehrt werden kann, wird in Kapitel 5.2.2.3 (vgl. D.3.13) dargestellt.

- **D.2.15**: Eine DoS-Attacke auf den Authentifikationsmanager kann neben der Schnittstelle über das SicAri-Environment (vgl. D.2.10) auch über die Web Service-Schnittstelle erfolgen. In Kapitel 5.2.2.4 wird auf diese Bedrohung (vgl. D.4.2) und die weiteren speziellen Web Service-Bedrohungen und deren Abwehr detailliert eingegangen.

- **D.2.16**: Die Sicherheitsrichtlinien sind in Form von XML-Dateien im Klartext gespeichert. Um eine Manipulation dieser zu vermeiden, ist eine Abspeicherung in verschlüsselter Form oder die Verwendung von digitalen Signaturen sinnvoll.

- **D.2.17**: Um über einen Handler Daten manipulieren zu können, muss der Angreifer einen vorhandenen Handler ersetzen oder einen neuen Handler installieren. Ohne Administrationsrechte ist dies nicht möglich.

- **D.2.18**: Das Abrufen von Security Tokens, über die öffentliche und statische Schnittstelle des AMH, ist durch eine Sicherheitsrichtlinie vor unerlaubten Zugriffen geschützt. Gelingt es einem Angreifer jedoch, die Sicherheitsrichtlinien zu umgehen (T.policy.*, T.secm.privilege-elevation), kann ein Angreifer diese Schnittstelle ausnutzen.

5.2.2.3 Authentifikationsmethoden einer Plattform

Durch die hier beschriebenen Abwehrmechanismen soll verhindert werden, dass sich ein Angreifer als regulärer Benutzer oder Administrator der Plattform maskieren kann.

Bedrohungen	Abwehr
T.pwd.eavesdropping	D.3.1
T.pwd.dictionary	D.3.2

Bedrohungen	Abwehr
T.softtoken.eavesdropping	D.3.3
T.softtoken.pwd.eavesdropping	D.3.1
T.softtoken.pwd.dictionary	D.3.2
T.softtoken.tampering	D.3.4
T.smartcard.eavesdropping	D.3.5 & D.3.3
T.smartcard.pin.eavesdropping	D.3.6 & D.3.1
T.smartcard.pin.dictionary	D.3.7
T.smartcard.tampering	D.3.4
T.st.eavesdropping	D.3.8
- T.st.invald	D.3.9
T.st.tampering	D.3.10 & D.3.4
T.st.runtime	D.3.11
T.auth.replay	D.3.12
T.auth.mitm	D.3.13
T.identity.spoofing	D.3.14
- T.repudiability	D.3.15
- T.privilege-elevation	D.3.15
- T.data-tampering	D.3.15
- T.data-eavesdropping	D.3.15

Tabelle 5.3: Zuordnung Bedrohungen und Abwehrmaßnahmen: Authentifikationsmethoden

- **D.3.1**: Das Ausspähen des Passworts bzw. der PIN durch Beobachten der Eingaben, kann durch technische Maßnahmen nicht verhindert werden. Die Benutzer des Systems können jedoch durch entsprechende Schulungen drauf hingewiesen werden, so dass eine sichere Passworteingabe erfolgt. Im Rahmen einer Schulung muss auch darauf hingewiesen werden, dass Passwörter im Allgemeinen nicht als Merkhilfe im Klartext in einer Datei abgespeichert werden sollten.

- **D.3.2**: Ein Angriff mittels der beschriebenen Dictionary-Attack (siehe Kapitel 5.1.4) kann durch einen verlangsamten Login-Prozess oder durch den Einsatz von CAPTCHAs verhindert werden. Da die Plattform mehrere Login-Module anbietet, ist der Login-Prozess in gewisser Weise schon durch das Durchgehen der einzelnen Login-Module verlangsamt.

- **D.3.3**: Um abzuwehren, dass ein Angreifer in den Besitz eines Soft Token gelangt, sind auf Seite der SicAri-Plattform keine technischen Maßnahmen möglich. Da das Soft Token auf einem Datenträger innerhalb eines PCs oder eines mobilen Endgeräts gespeichert ist, muss der Datenträger vor unerlaubten Zugriffen geschützt werden. Hier sind entsprechende Sicherheitsvorkehrungen des verwendeten Betriebssystems zu betrachten. Ein Angreifer kann natürlich auch diese überwinden. Wie dies verhindert werden kann, ist jedoch nicht Gegenstand dieser Arbeit.

- **D.3.4**: Um ein Soft Token bzw. eine Smart Card zu fälschen, benötigt der Angreifer detaillierte Kenntnisse über den Typ und die Eigenschaften der einzelnen Schlüssel, die für die Authentifikation benötigt werden. Außerdem müssen die auf dem Soft Token bzw. der Smart Card gespeicherten Zertifikate mit dem privaten Schlüssel der SicAri-CA signiert sein. Da dieser Schlüssel normalerweise nicht ohne Weiteres in den Besitz eines Angrei-

fers gelangen kann, ist eine Fälschung eines Soft Token bzw. einer Smart Card nahezu unmöglich.

- **D.3.5**: Der Versuch eines Angreifers an ein Soft Token bzw. eine Smart Card zu gelangen, kann durch technische Maßnahmen der SicAri-Plattfrom nicht abgewehrt werden. Durch entsprechende Schulungen kann der Benutzer auf die Wichtigkeit der sicheren Aufbewahrung der Smart Card hingewiesen werden. Kommt ein Angreifer dennoch in den Besitz einer Smart Card, so ist die für die Benutzung benötigte PIN der einzige Abwehrmechanismus.

- **D.3.6**: Um das Ausspähen der PIN zu verhindern, sind ähnliche Abwehrmechanismen wie in D.3.1 anzuwenden.

- **D.3.7**: Ein Dictionary-Angriff auf die PIN einer SmartCard ist in der Regel nicht möglich, da die SmartCard einen Zähler über die Anzahl der Fehlversuche besitzt. Nachdem eine kartenspezifische Maximalzahl (meist 3) erreicht ist, wird die Karte blockiert und kann nicht mehr verwendet werden. Erst durch eine Entsperrung durch den Kartenaussteller kann die Karte wieder verwendet werden. Dazu muss ein so genannter *PIN Unblocking Key (PUK)* eingegeben werden. Nach einer mehrmaligen falschen Eingabe des PUK kann die Karte endgültig nicht mehr verwendet werden.

- **D.3.8**: Um an ein ausgestelltes Security Token zu gelangen, hat der Angreifer entweder die Möglichkeit, das ST über einen unverschlüsselten Übertragungskanal abzuhören oder es von dem Gerät zu entwenden, auf dem das ST gespeichert ist. Da das ST innerhalb einer SicAri-Plattform im Rahmen des `Subject`-Objekts sicher verwahrt ist, besteht hier keine Möglichkeit an das ST zu gelangen. Authentifiziert sich ein Benutzer jedoch über ein Endgerät, das keine eigene Instanz der Plattform ausführt, so wird das ST ggf. auf dem Gerät gespeichert. Damit ein Angreifer nicht an das gespeicherte Security Token gelangen kann, sind ähnlich wie bei D.3.3, die Abwehrmechanismen des eingesetzten Betriebssystems die einzigen Möglichkeiten, dies zu verhindern. Ist ein Angreifer in den Besitz eines mobilen Endgeräts gelangt, so muss er sich oft nicht einmal bei dem Endgerät anmelden, um das Gerät nutzen zu können. Die Benutzerauthentifikation ist bei vielen Geräten aus Gründen der Benutzerfreundlichkeit oftmals deaktiviert und ein Angreifer kann so sehr leicht an ein ST gelangen.

  Ein Angreifer kann, wie bereits erwähnt, auch versuchen, ein Security Token über einen unverschlüsselten Übertragungskanal abzuhören. Um dies zu vermeiden, sollten entsprechende Sicherungstechniken ergriffen werden. Neben SSL/TLS oder IPSec kommen hier auch Web Service-Security-Mechanismen in Betracht. Jedoch ist letzteres mit Vorsicht zu verwenden. Auch wenn das ST mittels XMLenc verschlüsselt wird, kann der Angreifer das ST trotzdem abhören. Er kann sogar das verschlüsselte ST erfolgreich zur Authentifikation verwenden, wenn er sich bei der Plattform, für die die Verschlüsselung erstellt wurde, anmeldet.

  Da ein ST nur eine festgelegte Gültigkeit besitzt, kann ein Angreifer das ST nur für einen gewissen Zeitraum zur Maskierung verwenden (vgl. D.3.9).

- **D.3.9**: Ist ein Angreifer in den Besitz eines bereits abgelaufenen ST gelangt, kann er dieses nur erfolgreich zur Authentifikation verwenden, wenn er es schafft, die Systemzeit der Plattform zu verändern. Um dies zu verhindern, müssen die Sicherheitsmechansimen des eingesetzten Betriebssystems beansprucht werden.

- **D.3.10**: Die Fälschung eines ST ist, wie auch die Fälschung eines Soft Token oder einer Smart Card (vgl. D.3.4), schwierig, denn der Angreifer muss das ST mit einem gültigen Schlüssel eines als Administrator anerkannten Benutzers signieren. Erst dann wird es von einer Plattform zur Authentifikation akzeptiert. An diesen Schlüssel kommt der Angreifer nicht ohne weiteres (vgl. D.3.3 & D.3.5).

- **D.3.11**: Um zu vermeiden, dass ein Angreifer die Dienste der Plattform länger nutzt als die Gültigkeit des ST dies vorschreibt, muss die Plattform die Gültigkeit in bestimmten Zeitabschnitten oder bei jeder Aktion überprüfen.

- **D.3.12**: Um einen Replay-Angriff ausführen zu können, muss der Angreifer zunächst die unverschlüsselte Kommunikation mit dem Authentifikationmanager abhören und die entsprechenden Datenpakete mitschneiden. Der Angreifer kann nun die in Abhängigkeit der Authentifikationsmethode berechneten Authentifikationsanfrage einsehen. Da die Authentifikationsanfrage jedoch nur aus einem Hash-Wert ($M1$) bzw. mit dem privaten Schlüssel des Benutzers signierten Wert ($M2, M3$) besteht, kann der Angreifer nicht auf das verwendete Passwort, den verwendeten Schlüssel oder die beinhaltete Nonce schließen.

  Die Replay-Attack ist nicht möglich, da der Authentifikationsmanager diese erkennen kann. Versucht ein Angreifer ein abgehörtes Paket erneut an den Authentifikationsmanager zu senden, überprüft dieser, ob in seiner internen Datenbank eine Nonce für den angegebenen Benutzer existiert. Da dies nicht der Fall ist, wird er die Authentifikation mit einer Fehlermeldung abbrechen. Auch wenn der Angreifer zunächst eine neue Nonce für den Benutzer berechnen lässt, in dem er die Methode `getAuthenticationMethods` aufruft, gelingt die Replay-Attacke nicht. Beim Versuch das abgehörte Datenpaket einzuspielen erkennt der Authentifikationsmanager, dass die Authentifikationsanfrage nicht zu der vorhandenen Nonce passt.

- **D.3.13**: Bei der unverschlüsselten Verbindung mit dem Authentifikationsmanager kann ein Angreifer einen Man-in-the-Middle-Angriff vornehmen. Nachdem der Angreifer sich zwischen die Kommunikationspartner gesetzt hat, wartet er bis der Benutzer seine Authentifikationsanfrage stellt. Das zugehörige Datenpaket fängt der Angreifer ab und blockiert die weitere Verbindung zum Benutzer. Jetzt schickt er die Authentifikationsanfrage an den Authentifikationsmanager weiter und wird schließlich erfolgreich authentifiziert. Der Angreifer muss dies jedoch sehr schnell vornehmen, da ansonsten die erstellte Nonce abläuft und damit die Authentifikation fehlschlägt.

  Zur Abwehr solch eines Angriffs kann das SSL/TLS-Protokoll angewendet werden. Hierbei ist jedoch darauf zu achten, dass eine strikte Überprüfung der ausgetauschten Zertifikate erfolgt. Ansonsten kann ein Angreifer versuchen, dem Benutzer ein falsches Server-Zertifikat vorzuspiegeln und sich so als Authentifikationsmanager maskieren. Nur wenn das Zertifikat auf Vertrauenswürdigkeit überprüft wird, kann ein MITM-Angriff bei Verwendung des SSL/TLS-Protokolls vermieden werden.

- **D.3.14**: Um eine Identität maskieren zu können, muss sich ein Angreifer erfolgreich authentifizieren können. Dafür benötigt er entsprechende Authentifikationsdaten. Damit ein Angreifer diese nicht ohne Weiteres bekommen kann, existieren verschiedene Abwehrmechanismen (vgl. D.3.1 - D.3.12).

- **D.3.15**: Alle weiteren Bedrohungen, die sich durch das Maskieren einer Identität ergeben,

können von der Plattform nicht abgewehrt werden. Die Plattform kann selbst nicht unterscheiden, ob es sich um einen regulären Benutzer oder um einen Angreifer, der sich als regulärer Benutzer ausgibt, handelt. Die einzige Möglichkeit zur Abwehr besteht darin, eine Maskierung einer Identität zu verhindern (vgl. D.3.14).

5.2.2.4 Weitere Angriffspunkte

Bedrohungen	Abwehr
T.platform.spoofing	D.4.1
T.wsdl.data-eavesdropping	D.1.3
T.ws.dos	D.4.2
T.ws.malicious-content	D.4.3
T.ws.parameter-attacks	D.4.4
T.ws.unauthorized-access	D.4.5
T.ws.rekursive-references	D.4.6
T.ws.external-references	D.4.6
T.ws.protocol-threats	D.4.7
T.ws.security-tampering	D.4.8
T.ws.end-to-end-security	D.4.9

Tabelle 5.4: Zuordnung Bedrohungen und Abwehrmaßnahmen: Weitere Bedrohungen

- **D.4.1**: Für das erfolgreiche Maskieren einer Plattform bzw. das Einschleusen einer weiteren Plattform in eine SicAri-Infrastruktur, benötigt der Angreifer ein gültiges Administrator-Zertifikat, das von der SicAri-CA ausgestellt wurde. Nur dann sehen die anderen Plattformen die Plattform des Angreifers als vertrauenswürdig an und akzeptieren die Kommunikation mit dieser Plattform.

  An solch ein Zertifikat kann er durch die in Kapitel 5.2.1.4 dargestellten Methoden gelangen. In Kapitel 5.2.2.3 wird gezeigt, wie dies abgewehrt werden kann.

- **D.4.2**: Die Abwehr und Erkennung von DoS-Angriffen auf Web Services ist nicht einfach. Der Web Service kann selbst nicht entscheiden, ob es sich um einen DoS-Angriff handelt, denn er kann erst reagieren, wenn er bereits aufgerufen wurde.

  Sinnvoll ist deshalb die Installation eines Monitors, der die Zugriffe auf einen Web Service beobachtet. Dieser könnte in Axis in Form eines Handler implementiert werden, der die Anzahl der Zugriffe pro Zeiteinheit misst und analysiert, von woher eine Anfrage stammt. Beim Überschreiten eines festgelegten Werts kann der Handler, durch Erzeugen einer Fehlermeldung, den Zugriff auf den Web Service unterbinden.

  Um die Wahrscheinlichkeit für DoS-Angriffe mittels großer XML-Nachrichten abwehren zu können, ist eine Authentifikation der Benutzer sinnvoll. Dadurch ist ein Angriff zwar nicht ausgeschlossen, der Angreifer kann jedoch identifiziert werden. Auch hier ist ggf. eine Analyse der Anfrage und damit die Abwehr eines Angriffs über einen Axis-Handler möglich.

  Natürlich kann durch einen DoS-Angriff auch die Verfügbarkeit des Handler und damit auch von Axis selbst gestört werden. Hier kann durch Firewalls bzw. Paketfilter entgegengewirkt werden.

- **D.4.3**: Um Angriffe durch bösartigen Inhalt in XML-Dateien abwehren zu können, sollte eine XML-Firewall (siehe Kapitel 2.7.3) eingesetzt werden. Damit können diese Inhalte ggf. erkannt werden und entsprechende Nachrichten abgewiesen werden.

- **D.4.4**: Alle Angriffe auf dieser Ebene sind meist anwendungsspezifisch. Daher sollte die Anwendung vor der Verarbeitung die Parameter auf ihre Gültigkeit überprüfen. Auch durch eine XML-Firewall, die eine Nachricht gegen eine entsprechende Schema-Definition überprüft, können solche Angriffe erkannt und abgewehrt werden.

- **D.4.5**: Um den unerlaubten Zugriff auf die Web Services der SicAri-Plattform zu vermeiden, kann die Plattform so konfiguriert werden, dass sich der Benutzer zunächst authentifizieren muss. Dazu muss er ein vom Authentifikationsmanager zuvor ausgestellten ST vorzeigen. Die Bedrohung durch unerlaubte Zugriffe kann somit erfolgreich abgewehrt werden.

- **D.4.6**: Da Axis keine Unterstützung für XML-Entities mehr besitzt, können diese Bedrohungen abgewehrt werden.

- **D.4.7**: Alle Angriffe, die durch das jeweils eingesetzte Protokoll entstehen können, müssen durch entsprechende Maßnahmen, wie beispielsweise Firewalls bzw. Paketfilter, abgewehrt werden. Welche Maßnahmen hier zu ergreifen sind, muss entschieden werden, wenn feststeht, welche Transportprotokolle ein Web Service anbietet.

- **D.4.8**: Durch den Einsatz von sicheren Verschlüsselungs- und Signatur-Algorithmen und die Verwendung von ausreichend großen Schlüsseln, können Angriffe auf die eingesetzten XMLdsig- und XMLenc-Mechanismen abgewehrt werden.

- **D.4.9**: Der Einsatz des SSL/TLS-Protokolls im Web Service-Umfeld kann ggf. zur Realisierung einer Ende-zu-Ende-Sicherheit nicht eingesetzt werden, falls eine SOAP-Nachricht nicht direkt vom Sender zum Empfänger gesendet wird, sondern über verschiedene Zwischenknoten sein Ziel erreicht. Hier müssen daher die Mechanismen der Web Service-Security Spezifikation verwendet werden.

5.3 Risikoanalyse

Aus Zeitgründen konnte eine Risikoanalyse im Rahmen dieser Arbeit nicht mehr durchgeführt werden. Daher soll hier nur die grundsätzliche Vorgehensweise verdeutlicht werden.
Nachdem die Bedrohungen identifiziert wurden, müssen diese im Rahmen einer Risikoanalyse bewertet werden. Dabei wird für jede Bedrohung die Wahrscheinlichkeit für das Eintreten und der potenzielle Schaden, der dadurch verursacht werden kann, ermittelt.[21]
Das Risiko ergibt sich aus der Wahrscheinlichkeit multipliziert mit dem Schadenspotenzial. Für die Bewertung der Wahrscheinlichkeit und dem Schadenspotenzial bietet sich eine Skala von 1 bis 10 an. Wobei 1 für eine Bedrohung steht, die wahrscheinlich nie eintritt und 10 für eine Bedrohung, die fast sicher eintritt. Für das Schadenspotenzial steht der Wert 1 für einen minimalen Schaden und der Wert 10 für den größtmöglichen Schaden. Anhand der so entstehenden Skala von 1 bis 100 können die Entwickler und Betreiber einer SicAri-Plattform o die Bedrohungen mit dem größten Risiko und dem höchsten Schadenspotenzial ableiten und so gezielte Abwehrmechanismen ergreifen.

Dies sind jedoch nur zwei Kriterien. Oft ist es sinnvoll, weitere Kriterien festzulegen, um eine detailliertere Risikobewertung zu erhalten. Microsoft propagiert dafür das so genannte DREAD-Modell [64]. Das DREAD-Modell definiert fünf Kriterien. Für jede Bedrohung muss eine Bewertung anhand dieser Kriterien durchgeführt werden:

- *Damage potential* (Schadenspotenzial): Wie groß ist der Schaden, wenn die Sicherheitslücke ausgenutzt wird?
- *Reproducibility* (Reproduzierbarkeit): Wie leicht kann der Angriff reproduziert werden?
- *Exploitability* (Ausnutzbarkeit): Wie leicht kann ein Angriff gestartet werden?
- *Affected* (betroffene Benutzer): Wie viele Benutzer können ungefähr betroffen sein (in Prozent)?
- *Discoverability* (Auffindbarkeit): Wie leicht kann die Sicherheitslücke aufgefunden werden?

Durch eine Bewertung anhand eines einfachen Schemas, wie hoch (3), mittel (2) und gering (1), kann eine genauere Risikoanalyse erfolgen. Für jede Bedrohung wird anhand der fünf Kriterien festgelegt, ob diese mit hoch, mittel oder gering einzuschätzen sind. Danach werden die einzelnen Werte summiert. Die sich ergebende Skala von 5 bis 15 wird in drei Bereiche eingeteilt. Das Risiko für Bedrohungen mit einer Bewertung von 12 bis 15 ist als hoch einzustufen, 8 bis 11 stellt ein mittleres Risiko dar, und bei einem Wert von 5 bis 7 ist von einem eher geringen Risiko auszugehen.[64]
Um eine noch detailliertere Analyse vorzunehmen, kann es oft sinnvoll sein, noch weitere Kriterien festzulegen und diese ggf. unterschiedlich zu gewichten. Dies ist jedoch abhängig von dem jeweilig zu bewertenden System.

Kapitel 6

Evaluation

In diesem Kapitel wird, mit Hilfe verschiedener Testszenarien, die Leistungsfähigkeit der implementierten Funktionen untersucht.

6.1 Leistungsfähigkeitstests

Im Rahmen der im Folgenden beschriebenen Testszenarien soll die Leistungsfähigkeit der implementierten Funktionen untersucht werden. Hierbei wird die Authentifikation über den globalen Authentifikationsmanager und der SSO analysiert.
Die Tests umfassen jeweils mehrere Durchläufe mit unterschiedlich aktivierten Sicherheitsoptionen. Im ersten Fall wird gänzlich auf die Sicherheit durch das SSL/TLS-Protokoll bzw. WS-Security-Mechanismen verzichtet. Im zweiten Fall wird die Sicherheit durch das SSL/TLS-Protokoll gewährleistet und im letzten Fall durch die WS-Security-Mechanismen. Im Weiteren werden diese Fälle durch die Angabe der folgenden Schlüsselwörter unterschieden.

- **Ohne**: Keine Sicherheit
- **SSL**: Sicherheit durch das SSL/TLS-Protokoll
- **WSS**: Sicherheit durch WS-Security-Mechanismen

Testumgebung Die einzelnen Tests wurden auf einem Notebook mit den folgenden Eigenschaften durchgeführt.

- Intel Pentium M 750: 1.86 GHz
- 533 MHz FSB
- 1024 MB Arbeitsspeicher (Dual-Channel)

Die Verbindung zum Internet wurde über DSL mit den folgenden Geschwindigkeiten hergestellt.

- 3000 KBit/s Download
- 386 KBit/s Upload

Zielsetzung Die Tests sollen es ermöglichen, anhand der gewonnen Ergebnisse zu entscheiden, welche Optionen unter welchen Bedingungen am sinnvollsten einzusetzen sind.
Für die Authentifikation soll entschieden werden, welcher ST-Typ in Kombination mit welchen Sicherheitsoptionen im Hinblick auf die Leistungsfähigkeit am effizientesten ist.
Für den SSO soll neben der Leistungsfähigkeit des SSO auch das Antwortverhalten eines Web Service in Abhängigkeit der Nachrichtengröße und der eingesetzten Sicherheitsoptionen untersucht werden.
Für jeden Test werden die Gesamtlaufzeit und die Übertragungszeit sowie die Bearbeitungszeit der Axis-Handler berechnet. Anhand mehrerer Durchläufe werden schließlich die folgenden Werte bestimmt.

- Minimum
- Maximum
- Mittelwert/Durchschnitt
- Streuung

6.1.1 Authentifikation

Im Folgenden werden die Testszenarien für die globale Authentifikation und deren Ergebnisse vorgestellt.

6.1.1.1 Testszenario

Für dieses Testszenario wird zunächst, innerhalb der Entwicklungsumgebung Eclipse, eine Instanz einer SicAri-Plattform mit den folgenden Eigenschaften gestartet.

- Plattform sicari1:
  - Plattform-Administrator: sicari1
  - HTTP-Port: 8081
  - HTTPS-Port: 8881

Auf dieser Plattform wird beim Start automatisch der Authentifikationsmanager publiziert.

Um die globale Authentifikation eines Benutzers zu testen, wird eine zweite Instanz einer SicAri-Plattform mit den folgenden Eigenschaften gestartet.

- Plattform sicari2:
  - Plattform-Administrator: sicari2
  - HTTP-Port: 8082
  - HTTPS-Port: 8882

Nach dem Start der zweiten Plattform wird über ein Shell-Skript ein Benutzer über den Authentifikationsmanager authentifiziert. Dazu wird über die UDDI-Registry die entsprechende WSDL-Beschreibung abgerufen und daraus ein Client-Stub generiert. Dieser wird automatisch

im lokalen Environment publiziert, so dass dies bei den folgenden Authentifikationen nicht erneut durchgeführt werden muss.
Nachdem dies abgeschlossen ist, wird anhand eines Shell-Skripts wiederholt die Authentifikation eines Benutzers durchgeführt. Dies wird für die einzelnen Testszenarien jeweils 50 mal wiederholt.
Als Authentifikationsmethode wird das Soft Token-Verfahren verwendet. Für die anderen Authentifikationsmethoden sind vergleichbare Ergebnisse zu erwarten. Um eine Verzögerung durch die Eingabe des Pfades zum Soft Token sowie des Passworts durch den Benutzer zu vermeiden, wurden die Login-Module `JKSLoginModule` und `GlobalJKSLoginModule` verändert. Der Pfad zum Soft Token sowie das zugehörige Passwort wurden dazu fest einprogrammiert.
Insgesamt wird die Leistungsfähigkeit in Abhängigkeit von vier ST-Typen bestimmt:

1. ASN.1: ASN.1-basierendes Security Token
2. ASN.1 enc: verschlüsseltes ASN.1-basierendes Security Token
3. SAML: SAML-basierendes Security Token
4. SAML enc: verschlüsseltes SAML-basierendes Security Token

Für jedes ST wird jeweils mit den zuvor definierten Sicherheitsoptionen getestet. Dadurch ergeben sich insgesamt 12 Testszenarien. Bei 50 Testläufen pro Testszenario sind dies insgesamt 600 Testläufe.

6.1.1.2 Messpunkte

Zur Berechnung der Gesamtlaufzeit, Übertragungszeit sowie der Bearbeitungszeit des AMH wurden an verschiedenen Punkten in der Nachrichtenverarbeitung Messungen vorgenommen. Bei jeder Messung wird die aktuelle Systemzeit über das *Logging*-System der Plattform in eine Datei geschrieben.
Die einzelnen Messpunkte sind in Abbildung 6.1 dargestellt (grüne Markierungen).
Die folgende Liste gibt einen Überblick über die Position der Messpunkte und die einzelnen Messungen. Insgesamt werden bei einer Authentifikation 22 Messwerte bestimmt.

A1. `AbstractLoginModule.login()`: Start

A2. `GlobalJKSLoginModule.authenticate()`:
vor `AuthenticationManagerWS.getAuthenticationMethods()`

A3. AMH: Axis-Client-Engine, Request-Flow: Start

A4. AMH: Axis-Client-Engine, Request-Flow: Ende

A5. AMH: Axis-Server-Engine, Request-Flow: Start

A6. AMH: Axis-Server-Engine, Request-Flow: Ende

A7. AMH: Axis-Server-Engine, Response-Flow: Start

A8. AMH: Axis-Server-Engine, Response-Flow: Ende

A9. AMH: Axis-Server-Engine, Response-Flow: Start

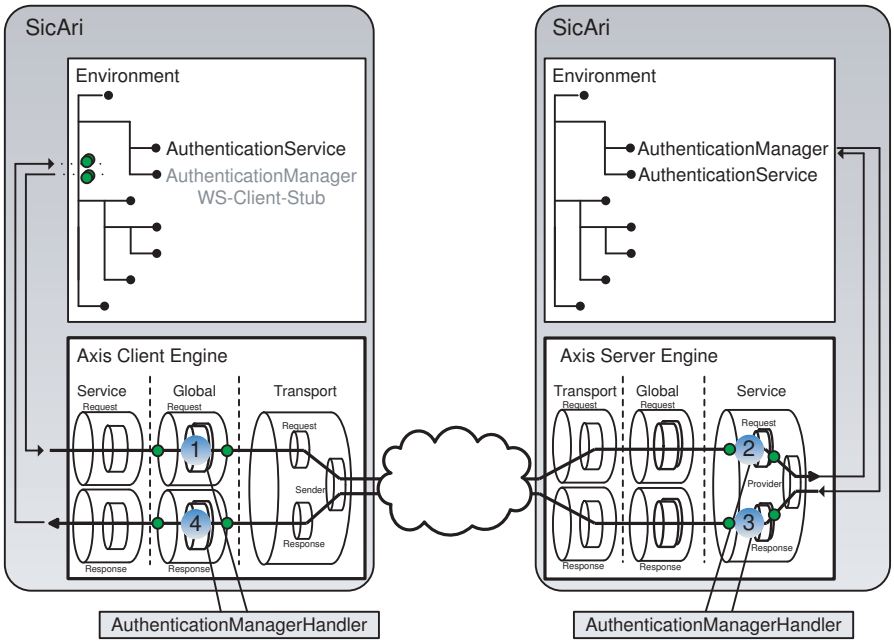

Abbildung 6.1: Messpunkte im Nachrichtenfluss der Authentifikation

A10. AMH: Axis-Server-Engine, Response-Flow: Ende

A11. `GlobalJKSLoginModule.authenticate()`: nach `AuthenticationManagerWS.getAuthenticationMethods`

A12. `GlobalJKSLoginModule.authenticate()`: vor `AuthenticationManagerWS.authenticate()`

A13. AMH: Axis-Client-Engine, Request-Flow: Start

A14. AMH: Axis-Client-Engine, Request-Flow: Ende

A15. AMH: Axis-Server-Engine, Request-Flow: Start

A16. AMH: Axis-Server-Engine, Request-Flow: Ende

A17. AMH: Axis-Server-Engine, Response-Flow: Start

A18. AMH: Axis-Server-Engine, Response-Flow: Ende

A19. AMH: Axis-Server-Engine, Response-Flow: Start

A20. AMH: Axis-Server-Engine, Response-Flow: Ende

A21. `GlobalJKSLoginModule.authenticate()`: nach `AuthenticationManagerWS.authenticate()`

A22. `AbstractLoginModule.commit()`: Ende

Anhand dieser Messwerte werden die folgenden Ergebnisse berechnet:

B.1 **`getAuthenticationMethods`**:

B.1.1. Bearbeitungszeit Axis-Client-Engine, Request-Flow = A4 - A3

B.1.2. Übertragungszeit Anfrage = A5 - A4

B.1.3. Bearbeitungszeit Axis-Server-Engine, Request-Flow = A6 - A5

B.1.4. Bearbeitungszeit Authentifikationsmanager = A7 - A6

B.1.5. Bearbeitungszeit Axis-Server-Engine, Response-Flow = A8 - A7

B.1.6. Übertragungszeit Antwort = A9 - A8

B.1.7. Bearbeitungszeit Axis-Client-Engine, Response-Flow = A10 - A9

B.2 **`authenticate`**:

B.2.1. Bearbeitungszeit Axis-Client-Engine, Request-Flow = A14 - A13

B.2.2. Übertragungszeit Anfrage = A15 - A14

B.2.3. Bearbeitungszeit Axis-Server-Engine, Request-Flow = A16 - A15

B.2.4. Bearbeitungszeit Authentifikationsmanager = A17 - A16

B.2.5. Bearbeitungszeit Axis-Server-Engine, Response-Flow = A18 - A17

B.2.6. Übertragungszeit Antwort = A19 - A18

B.2.7. Bearbeitungszeit Axis-Client-Engine, Response-Flow = A20 - A19

B.3 Gesamtlaufzeit = A22 - A1

B.4 Bearbeitungszeit AMH = B.1.1 + B.1.3 + B.1.5 + B.1.7 + B.2.1 + B.2.3 + B.2.5 + B.2.7

B.5 Gesamte Übertragungszeit = B.1.2 + B.1.6 + B.2.2 + B.2.6

B.6 Bearbeitungszeit Authentifikationsmanager = B.1.4 + B.2.4

6.1.1.3 Ergebnisse

Die Grundlage der im Folgenden beschriebenen Ergebnisse sind die Messwerte der einzelnen Testdurchläufe. Diese sind im Verzeichnis `test-protocols/authentication` auf der beiliegenden CD abgelegt.

Bearbeitungszeit Authentifikationsmanager Die Bearbeitungszeit des Authentifikationsmanagers wird von den unterschiedlichen Testszenarien nicht beeinflusst. Das bedeutet, dass für jedes Testszenario die Bearbeitungszeit annähernd gleich ist.
Anhand der einzelnen Testläufe wurden die folgenden Werte berechnet:

- Minimum: 375 msec
- Maximum: 1141 msec
- Mittelwert: 513 msec
- Streuung: 202 msec

Zusammengefasst benötigt der Authentifikationsmanager zur Bearbeitung der zwei Methoden Aufrufe `getAuthenticationMethods()` und `authenticate()` durchschnittlich 513 msec.

Gesamtlaufzeit Die in Tabelle 6.1 dargestellten Werte zeigen die Ergebnisse der einzelnen Testszenarien für die Gesamtlaufzeit. Abbildung 6.2 zeigt die durchschnittliche Gesamtlaufzeit als Balkendiagramm.
Anhand der dargestellten Ergebnisse lässt sich ablesen, dass für die Authentifikation durchschnittlich 3,5 Sekunden benötigt werden. Dies ist unabhängig davon, ob die Sicherheit durch SSL/TLS gewährleistet wird oder keine Sicherheitsoption verwendet wird. Wird allerdings die Kommunikation mittels WS-Security-Mechanismen abgesichert, erhöht sich die Gesamtlaufzeit auf durchschnittlich 6 Sekunden. Auf die Gründe für diesen Anstieg wird im nächsten Abschnitt eingegangen.
Wird bei der Übertragung das ST zusätzlich verschlüsselt, so erhöht sich die Gesamtlaufzeit beim Einsatz eines ANS.1-ST geringfügig. Wird ein SAML-ST ausgestellt, so erhöht sich die Gesamtlaufzeit um ca. 500 msec. Im nächsten Abschnitt wird dieser Unterschied erklärt.

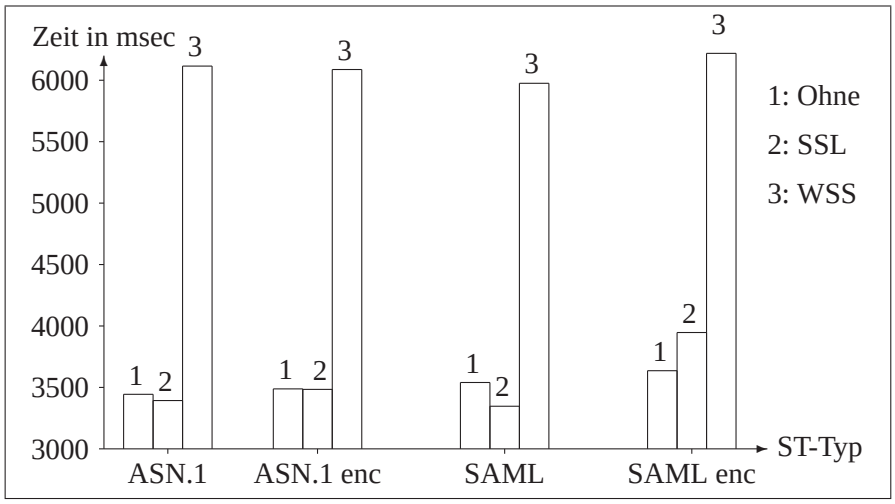

Abbildung 6.2: Durchschnittliche Gesamtlaufzeit: Authentifikation

ST	Sicherheit	Minimum	Maximum	Mittelwert	Streuung
ASN.1 -ST	ohne	2781	3985	3444	272
	SSL	3234	4312	3393	195
	WSS	5687	7625	6115	376
enc ASN.1 -ST	ohne	2891	5344	3488	359
	SSL	2844	4641	3484	262
	WSS	5781	7750	6087	353
SAML -ST	ohne	2766	4609	3540	323
	SSL	3156	4719	3347	245
	WSS	5640	6907	5975	308
enc SAML -ST	ohne	3406	5047	3636	301
	SSL	3219	4891	3947	422
	WSS	5813	7796	6219	447

Tabelle 6.1: Ergebnisse Soft Token-Authentifikation: Gesamtlaufzeit

Bearbeitungszeit AMH Die in Tabelle 6.2 dargestellten Werte zeigen die Ergebnisse der einzelnen Testszenarien für die Bearbeitungszeit des Axis-Handler AMH. Abbildung 6.3 zeigt die durchschnittliche Bearbeitungszeit als Balkendiagramm.
Anhand der Ergebnisse lässt sich ableiten, dass der AMH für die Bearbeitung der einzelnen Nachrichten durchschnittlich 1 Sekunde benötigt. Die Verschlüsselung des ASN.1-ST erhöht die Bearbeitungszeit kaum. Für die Verschlüsselung des SAML-ST erhöht sich Bearbeitungszeit auf ca. 1,5 Sekunden. Die unterschiedlichen Verschlüsselungsalgorithmen und Schlüssellängen könnten dies evtl. erklären. Für die Verschlüsselung des ASN.1-ST wird der Algorithmus 3DES (Schlüssellänge 168 Bit) eingesetzt, für die Verschlüsselung des SAML-ST der Algorithmus AES mit einer Schlüssellänge von 128 Bit. Für diese erhöhte Bearbeitungszeit spricht jedoch vielmehr die Tatsache, dass die Verschlüsselung des SAML-ST auf Basis der DOM-Darstellung erfolgt, die Verschlüsselung des ASN.1-ST dagegen auf Basis der binären Darstellung des ST.
Werden zur Sicherung der Vertraulichkeit und Integrität der SOAP-Nachricht WS-Security-Mechanismen eingesetzt, erhöht sich die Bearbeitungszeit außerdem auf durchschnittlich 3,3 Sekunden. Die DOM-basierte Verschlüsselung und Signatur des SOAP-Body ist sehr aufwändig und erklärt diesen Anstieg der Bearbeitungszeit.

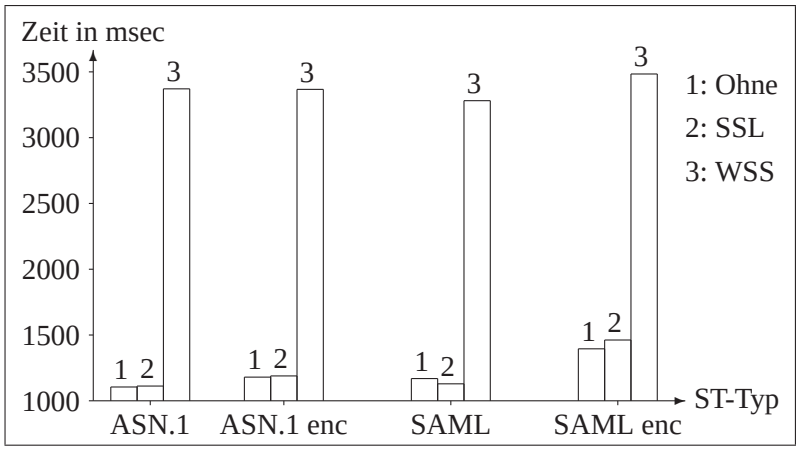

Abbildung 6.3: Durchschnittliche Bearbeitungszeit: AMH

ST	Sicherheit	Minimum	Maximum	Mittelwert	Streuung
ASN.1 -ST	ohne	669	1265	1105	82
	SSL	1062	1158	1112	27
	WSS	3078	4875	3371	326
enc ASN.1 -ST	ohne	673	1906	1180	179
	SSL	655	1346	1189	89
	WSS	3169	4358	3367	237
SAML -ST	ohne	671	1641	1169	144
	SSL	1048	1313	1129	48
	WSS	3095	4234	3281	229
enc SAML -ST	ohne	1295	2124	1395	109
	SSL	704	2561	1462	234
	WSS	3202	5141	3484	405

Tabelle 6.2: Ergebnisse Soft Token-Authentifikation: Bearbeitungszeit Axis-Handler AMH

Übertragungszeit Die in Tabelle 6.3 dargestellten Werte zeigen die Ergebnisse der einzelnen Testszenarien für die Übertragungszeit. Abbildung 6.4 zeigt die durchschnittlich beanspruchte Übertragungszeit als Balkendiagramm.
Es wird deutlich, dass die Übertragung der einzelnen Nachrichten bei der Authentifikation durchschnittlich 1 Sekunde beansprucht. Dies ist unabhängig von dem ausgestellten ST. Werden zur Gewährleistung der Vertraulichkeit und Integrität der Nachrichten WS-Security-Mechanismen eingesetzt, so erhöht sich die Übertragungszeit auf durchschnittlich 1,5 Sekunden. Dies ist dadurch zu erklären, dass sich durch den Einsatz der WS-Security-Mechanismen die Größe der ausgetauschten Nachrichten etwa um den Faktor 5 vergrößert.

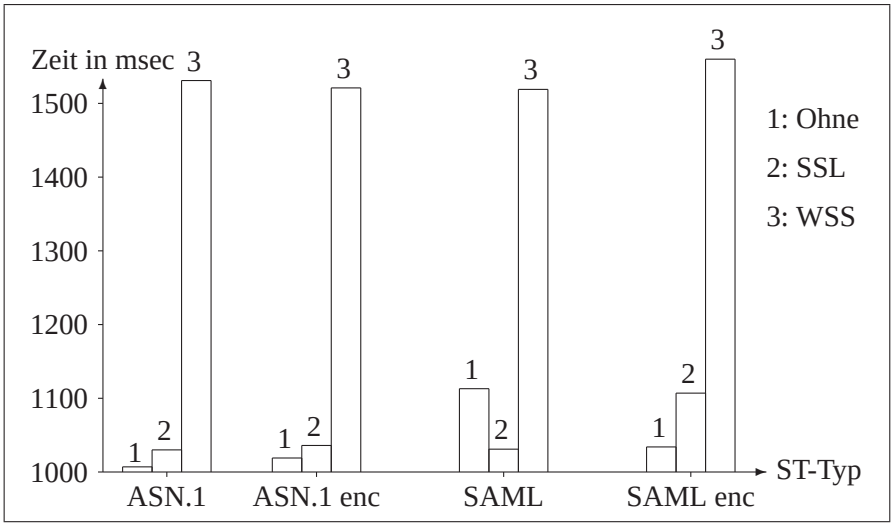

Abbildung 6.4: Durchschnittliche Übertragungszeit: Authentifikation

ST	Sicherheit	Minimum	Maximum	Mittelwert	Streuung
ASN.1 -ST	ohne	937	1187	1007	60
	SSL	954	1249	1030	52
	WSS	1408	1813	1531	107
enc ASN.1 -ST	ohne	951	1173	1019	55
	SSL	969	1187	1036	54
	WSS	1358	1704	1521	80
SAML -ST	ohne	953	1673	1113	139
	SSL	954	1188	1031	52
	WSS	1374	1781	1519	83
enc SAML -ST	ohne	967	1217	1034	64
	SSL	954	1345	1107	95
	WSS	1405	1859	1560	88

Tabelle 6.3: Ergebnisse Soft Token-Authentifikation: Übertragungszeit

6.1.1.4 Fazit

Die Ergebnisse zeigen, dass der Typ des ausgestellten Security Tokens für die Leistungsfähigkeit der Authentifikation vernachlässigt werden kann. Es werden bei jedem Typ ca. 3,5 Sekunden benötigt, unabhängig davon, ob die Sicherheit durch das SSL/TLS-Protokoll gewährleistet wird oder keine Sicherheitsoptionen aktiviert sind. Wird die verschlüsselte Form eines ST gewählt, so erhöht das die Laufzeit leicht. In der Praxis bringt jedoch die verschlüsselte Form des ST kaum Vorteile (siehe Kapitel 5.2.2.3, Abwehrmechanismen D.3.8)
Der Einsatz der WSS-Security-Mechanismen erhöht die Gesamtlaufzeit um etwa das Doppelte. In der Praxis ist es daher nicht sinnvoll, diese Sicherheitsoption zu verwenden, zumal die Authentifikation immer nur zwischen zwei Endpunkten erfolgt und so die Punkt-zu-Punkt-Sicherheit ausreichend ist.

6.1.2 SSO

Im Folgenden werden die Testszenarien für den SSO und deren Ergebnisse vorgestellt.

6.1.2.1 Testszenario

Für die Untersuchung des SSO wurde der Web Service `HelloTest` implementiert. Dieser bietet die Methode `getString()` an. Diese Methode liefert einen `String` einer festgelegten Länge zurück. Durch entsprechende Konfiguration dieses Web Services werden `Strings` in der Größe 1 KB, 10 KB, 100 KB, 500 KB und 1024 KB erzeugt und an den Anfragenden gesendet.
Für dieses Testszenario wird zunächst, innerhalb der Entwicklungsumgebung Eclipse, eine Instanz einer SicAri-Plattform mit den folgenden Eigenschaften gestartet.

- Plattform sicari2
  - Plattform-Administrator: sicari2
  - HTTP-Port: 8082
  - HTTPS-Port: 8882

Nach dem Start wird der HelloTest-WS installiert und veröffentlicht.

Um den SSO unter Aufruf des HelloTest-WS zu testen, wurde eine zweite Instanz einer SicAri-Plattform mit den folgenden Eigenschaften gestartet.

- Plattform sicari1
  - Plattform-Administrator: sicari1
  - HTTP-Port: 8081
  - HTTPS-Port: 8881

Nach dem Start dieser Plattform wird in einer Schleife der HelloTest-WS aufgerufen. Dazu wird dieser zunächst einmal aufgerufen und anschließend der zugehörige WS-Client-Stub im lokalen Environment publiziert. Für alle weiteren Aufrufe dieses WS entfällt somit die Suche in der UDDI-Registry und die Generierung des WS-Client-Stub aus der WSDL-Beschreibung.

Danach wird der WS 200 mal aufgerufen. Dabei wird für den SSO bei jedem Aufruf das ASN.1-ST des Benutzers zur entfernten Plattform übertragen und der Benutzer dort auf Basis dieses ST authentifiziert.
Insgesamt wird die Leistungsfähigkeit in Abhängigkeit von 5 Antwort-Nachrichtengrößen bestimmt. In Abhängigkeit der eingesetzten Sicherheitsoptionen entsteht eine *Overhead* durch den SOAP-Envelope.

Nachrichtengröße	Overhead: Ohne, SSL	Overhead: WSS
1 KB	510 Byte (51,00%)	5.529 Byte (552,90%)
10 KB	510 Byte (5,10%)	8.680 Byte (86,80%)
100 KB	510 Byte (0,51%)	40.256 Byte (40,36%)
500 KB	510 Byte (0,10%)	180.612 Byte (36,12%)
1024 KB	510 Byte (0,05%)	356.047 Byte (35,61%)

Tabelle 6.4: Antwortnachrichtengröße und Overhead durch SOAP

Die Größe der Anfragenachricht beträgt bei jedem Test 2.029 Byte und beeinflusst damit die Vergleichbarkeit der einzelnen Testszenarien nicht. Sie hat jedoch einen Einfluss auf die Gesamtlaufzeit, Übertragungszeit und Bearbeitungszeit. Wenn in den im Folgenden dargestellten Diagrammen beispielsweise eine Nachrichtengröße von 100 KB angezeigt ist, so sind dies, wenn der Overhead durch den SOAP-Envelope sowie die Größe der Anfragenachricht hinzu addiert wird, eigentlich 104.939 Byte bzw. 102,48 KB.
Zusätzlich wird mit einer festgelegten Nachrichtengröße von 10 KB auch 50 mal der WS aufgerufen, ohne dass hier der WS-Client-Stub im lokalen Environment publiziert wird. Damit soll die Leistungsfähigkeit für den Abruf der WSDL-Beschreibung und die Generierung des WS-Client-Stubs untersucht werden.

6.1.2.2 Messpunkte

Zur Berechnung der Gesamtlaufzeit, Übertragungszeit, der Bearbeitungszeit des AMH sowie des Zeitbedarfs für den SSO wurden an verschiedenen Punkten in der Nachrichtenverarbeitung Messungen vorgenommen. Bei jeder Messung wird die aktuelle Systemzeit über das *Logging*-System der Plattform in eine Datei geschrieben.
Die einzelnen Messpunkte sind in Abbildung 6.5 dargestellt (grüne Markierungen).
Die folgende Liste gibt einen Überblick über die Position der Messpunkte und die einzelnen Messungen. Insgesamt werden bei einem Aufruf 10 Messwerte bestimmt.

A.1 `HelloTest`-WS: Anfrage

A.2 STH: Axis-Client-Engine, Request-Flow: Start

A.3 STH: Axis-Client-Engine, Request-Flow: Ende

A.4 STH: Axis-Server-Engine, Request-Flow: Start

A.5 STH: Axis-Server-Engine, Request-Flow: Ende

A.6 STH: Axis-Server-Engine, Response-Flow: Start

A.7 STH: Axis-Server-Engine, Response-Flow: Ende

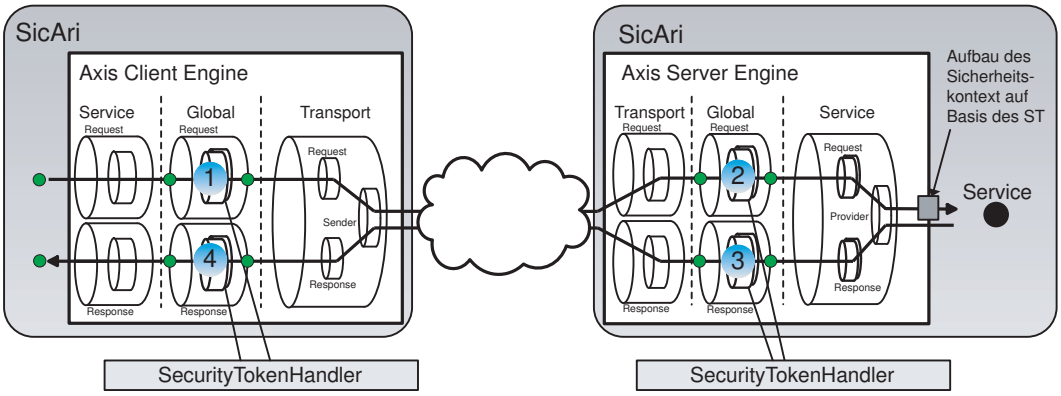

Abbildung 6.5: Messpunkte im Nachrichtenfluss des SSO

A.8 STH: Axis-Client-Engine, Response-Flow: Start

A.9 STH: Axis-Client-Engine, Response-Flow: Ende

A.10 `HelloTest`-WS: Antwort

Anhand dieser Messwerte werden die folgenden Ergebnisse berechnet:

B.1 Bearbeitungszeit Axis-Client-Engine, Request-Flow = A3 - A2

B.2 Übertragungszeit Anfrage = A4 - A3

B.3 Bearbeitungszeit Axis-Server-Engine, Request-Flow = A5 - A4

B.4 Bearbeitungszeit `HelloTest` = A6 - A5

B.5 Bearbeitungszeit Axis-Server-Engine, Response-Flow = A7 - A6

B.6 Übertragungszeit Antwort = A8 - A7

B.7 Bearbeitungszeit Axis-Client-Engine, Request-Flow = A9 - A8

B.8 Gesamtlaufzeit = A10 - A1

B.9 Bearbeitungszeit STH = B.1 + B.3 + B.5 + B.7

B.10 Gesamte Übertragungszeit = B.2 + B.6

B.11 Zeitbedarf SSO = B.1 + B.2 + B.3

6.1.2.3 Ergebnisse

Die Grundlage der im Folgenden beschriebenen Ergebnisse, sind die Messwerte der einzelnen Testdurchläufe. Diese sind im Verzeichnis `test-protocols/sso` auf der beiliegenden CD abgelegt.

SSO Die Zeit, die für den SSO benötigt wird, wird von der in diesem Test variierten Nachrichtengröße nicht beeinflusst, da sich diese nur auf die Größe der Antwortnachricht auswirkt. Der Zeitbedarf ist jedoch abhängig von der Größe der Anfragenachricht. Wird beispielsweise ein Web Service aufgerufen, der ein sehr großes XML-Dokument als Eingabeparameter erwartet, so wirkt sich dies auf die Übertragungszeit aus. Wenn dieses Dokument durch WS-Security-Mechanismen verschlüsselt wird, so kostet dies entsprechend mehr Zeit als in dem hier gewählten Beispiel.
Der Zeitbedarf für den SSO ergibt sich aus der Bearbeitungszeit der Anfrage auf Client- und Server-Seite sowie der benötigten Übertragungszeit.
Anhand der einzelnen Testläufe wurden die in Tabelle 6.5 dargestellten Werte bestimmt:

Sicherheit	Minimum	Maximum	Mittelwert	Streuung
ohne	500	703	547	34
SSL	782	546	589	39
WSS	968	1625	1038	63

Tabelle 6.5: Bearbeitungszeit SSO

Es wird deutlich, dass für den SSO ohne Sicherheitsoptionen durchschnittlich 547 msec benötigt werden. Wird die Kommunikation mittels des SSL/TLS-Protokolls abgesichert, so erhöht sich der Zeitbedarf auf annähernd 600 msec. Wird die Anfragenachricht mittels der WS-Security-Meachnismen verschlüsselt und signiert, so steigt der Zeitbedarf auf durchschnittlich 1 Sekunde an.

Gesamtlaufzeit Die in Tabelle 6.6 dargestellten Werte zeigen die Ergebnisse der einzelnen Testszenarien für die Gesamtlaufzeit. Abbildung 6.6 zeigt die durchschnittliche Gesamtlaufzeit als Diagramm.

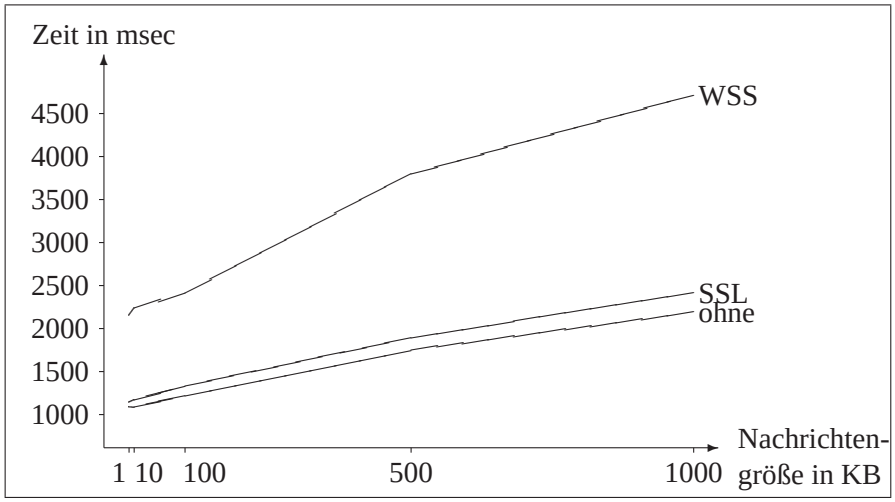

Abbildung 6.6: Durchschnittliche Gesamtlaufzeit: SSO

Die Ergebnisse zeigen, dass die Gesamtlaufzeit mit der Nachrichtengröße ansteigt. Bei einer Größe von 1 und 10 KB wird der Unterschied noch nicht deutlich, hier wird durchschnittlich jeweils eine Sekunde benötigt. Die Aktivierung der WS-Security-Mechanismen erhöht die Lauf-

zeit auf ca. 2 Sekunden. Bei einer Nachrichtengröße von 1000 KB steigt die Gesamtlaufzeit auf durchschnittlich 2,2 (Ohne), 2,4 (SSL) bzw. 4,7 Sekunden (WSS).
Insgesamt wird die Gesamtlaufzeit, durch den Einsatz der WS-Security-Mechanismen, auf etwas mehr als das Doppelte erhöht.
Wird der generierte WS-Client-Stub nicht im lokalen Environment publiziert, so erhöht sich die Gesamtlaufzeit erheblich. In Abhängigkeit der aktivierten Sicherheitsoptionen werden jetzt durchschnittlich 8 - 10 Sekunden benötigt. Wird dieser Wert mit den anderen Testwerten verglichen, so ist erkennbar, dass für den Abruf der WSDL-Beschreibung aus der UDDI-Registry und die Generierung des WS-Client-Stub ca. 7 - 8 Sekunden benötigt werden. Die Steigerung von 1 Sekunde bei Verwendung der SSL/TLS-Verbindung ist dadurch zu erklären, dass nicht nur zur Plattform, auf der der Web-Service installiert ist, eine SSL/TLS-Verbindung hergestellt wird, sondern auch zur UDDI-Registry.

ST	Sicherheit	Minimum	Maximum	Mittelwert	Streuung
1 KB	ohne	1031	1266	1091	42
	SSL	1094	1297	1146	42
	WSS	2078	2484	2158	80
10 KB	ohne	1031	1329	1085	44
	SSL	1094	2437	1173	127
	WSS	2078	6547	2239	394
10 KB (neuer WS-Stub)	ohne	8016	9703	8218	245
	SSL	8969	11422	9250	426
	WSS	10438	15953	10881	985
100 KB	ohne	1094	2172	1214	109
	SSL	1171	5797	1334	363
	WSS	2203	3062	2418	133
500 KB	ohne	1656	2078	1745	62
	SSL	1782	5359	1894	256
	WSS	3656	4812	3802	150
1 MB	ohne	2062	2390	2193	77
	SSL	2125	6906	2416	534
	WSS	4343	6703	4715	298

Tabelle 6.6: Ergebnisse SSO: Gesamtlaufzeit

Bearbeitungszeit STH Die in Tabelle 6.7 dargestellten Werte zeigen die Ergebnisse der einzelnen Testszenarien für die Bearbeitungszeit im Axis-Handler STH. Abbildung 6.7 zeigt die durchschnittliche Bearbeitungszeit als Diagramm.
Die Ergebisse zeigen, dass für die Bearbeitungszeit durchschnittlich ca. 450 msec benötigt werden, wenn ohne Sicherheitsoptionen oder mit aktivierten SSL/TLS-Protokoll getestet wurde. Bei SSL/TLS ist die Bearbeitungszeit etwas höher, was auf die Validierung der SSL/TLS-Verbindung zurückzuführen ist, die im STH vorgenommen wird.
Werden die WS-Security-Mechanismen aktiviert, so steigt die Bearbeitungszeit in Abhängigkeit der Nachrichtengröße auf 1,5 - 2,6 Sekunden an.

ST	Sicherheit	Minimum	Maximum	Mittelwert	Streuung
1 KB	ohne	375	577	429	27
	SSL	437	625	473	28
	WSS	1390	1782	1459	58
10 KB	ohne	390	657	423	25
	SSL	437	640	474	27
	WSS	1390	5861	1516	363
10 KB (neuer WS-Stub)	ohne	438	625	486	29
	SSL	453	1016	498	80
	WSS	2422	4797	2647	319
100 KB	ohne	390	812	438	40
	SSL	438	687	485	37
	WSS	1437	2171	1556	111
500 KB	ohne	374	563	424	20
	SSL	437	610	474	24
	WSS	2156	3031	2282	108
1 MB	ohne	374	485	422	17
	SSL	437	4000	520	367
	WSS	2405	4249	2616	231

Tabelle 6.7: Ergebnisse SSO: Bearbeitungszeit Axis-Handler STH

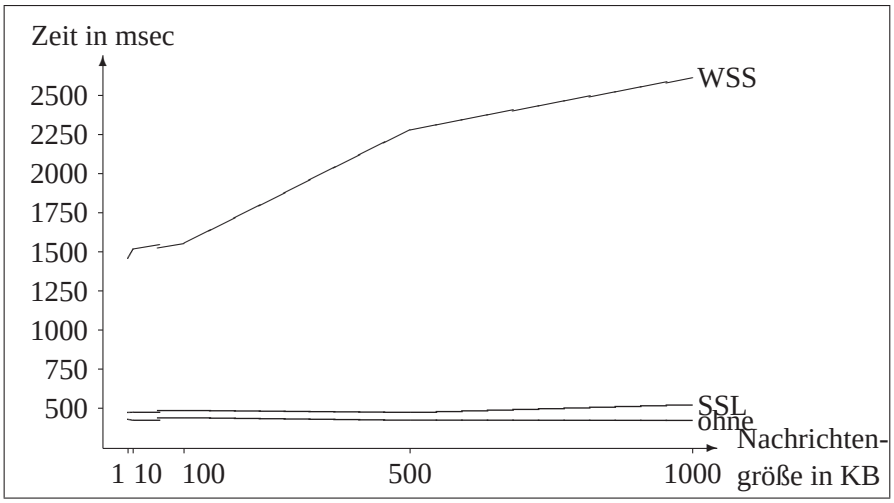

Abbildung 6.7: Durchschnittliche Bearbeitungszeit: STH

Übertragungszeit Die in Tabelle 6.8 dargestellten Werte zeigen die Ergebnisse der einzelnen Testszenarien für die Übertragungszeit. Abbildung 6.8 zeigt die durchschnittliche Übertragungszeit als Diagramm.
Wird das SSL/TLS-Protokoll aktiviert, so steigt die Übertragungszeit etwas an. Dies ist auf den zusätzlichen Zeitbedarf für den Aufbau der SSL/TLS-Verbindung zurückzuführen. Werden die WS-Security-Mechanismen zur Gewährleistung der Kommunikationsicherheit verwendet, so steigt die Übertragungszeit weiter an. Dies ist dadurch zu erklären, dass die Verschlüsselung und Signatur der Nachrichten deren Gesamtgröße erhöhen.

ST	Sicherheit	Minimum	Maximum	Mittelwert	Streuung
1 KB	ohne	313	516	352	27
	SSL	359	532	412	61
	WSS	343	547	384	36
10 KB	ohne	327	499	350	28
	SSL	327	1688	378	102
	WSS	343	2452	405	152
10 KB (neuer WS-Stub)	ohne	359	422	383	16
	SSL	359	453	394	20
	WSS	406	828	596	76
100 KB	ohne	344	798	418	68
	SSL	360	4703	461	308
	WSS	389	749	492	82
500 KB	ohne	703	797	744	17
	SSL	750	4172	827	239
	WSS	891	1516	1047	54
1 MB	ohne	860	1062	910	22
	SSL	984	1172	1038	30
	WSS	984	1532	1153	103

Tabelle 6.8: Ergebnisse SSO: Übertragungszeit

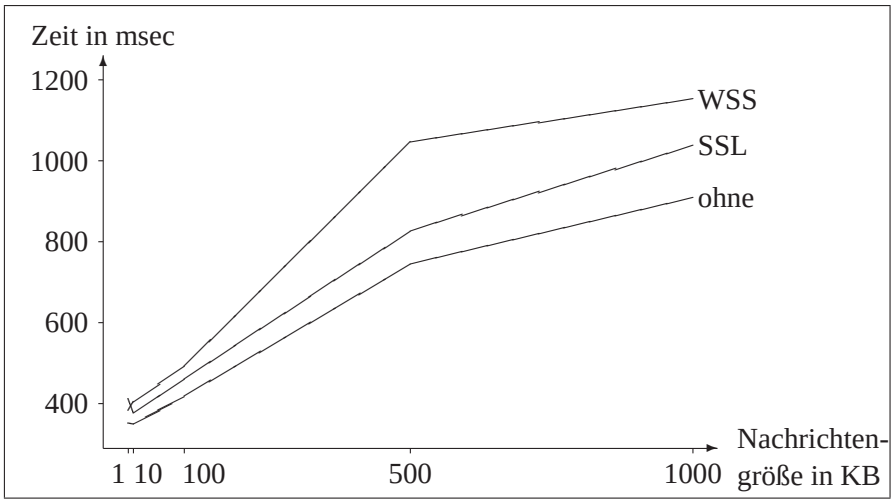

Abbildung 6.8: Durchschnittliche Übertragungszeit: SSO

6.1.2.4 Fazit

Zusammenfassend lässt sich aussagen, dass die WS-Security-Mechanismen den Aufruf eines WS auf einer entfernten Plattform erheblich verzögern. In der Praxis sollten diese Sicherheitsmechanismen daher möglichst nur dann eingesetzt werden, wenn die Laufzeit keine besondere Rolle spielt. Wird eine Nachricht über mehrere Zwischenknoten geleitet, ist es zur Wahrung der Integrität und Vertraulichkeit im Sinn einer Ende-zu-Ende-Sicherheit zwingend erforderlich, WS-Security-Mechanismen einzusetzen.

Da sich die Verwendung des SSL/TLS-Protokolls auf die Laufzeit kaum auswirkt, sollte dieses Protokoll immer eingesetzt werden. Ohne eine Sicherung der Vertraulichkeit und Integrität könnte ein Angreifer die Kommunikation abhören und so in den Besitz eines Security Token gelangen (siehe Kapitel 5).

Kapitel 7

Abschlussbetrachtung

7.1 Fazit

Die Arbeit hat gezeigt, dass die Sicherheitsziele Authentizität, Integrität oder Vertraulichkeit im Rahmen des WS-Umfelds erfüllt werden können. Durch die Erstellung von Verschlüsselungen und Signaturen können SOAP-Nachrichten vor unerwünschten Einblicken und Manipulationen geschützt werden. Die Authentisierung durch die verschiedenen Mechanismen der Plattform gewährleistet die Authentizität des sich damit identifizierenden Benutzers. Die Erweiterung der Authentifikationsmechanismen ermöglicht es, die Benutzer durch eine zentrale Komponente über ein auf Web Services basierendes Protokoll zu authentisieren. Die ursprüngliche Protokollspezifikation für die globale Authentifikation musste dafür teilweise auf die Bedingungen von Web Services angepasst werden. Der implementierte Authentifikationsmanager ist ein neuer Vertrauensanker in der SicAri-Infrastruktur und trägt maßgeblich zur Sicherheit bei.

Durch die Übertragung eines Security Token, beim Aufruf eines Web Service einer entfernten Plattform und die Validierung dieses Security Token auf der entfernten Plattform, konnte der SSO innerhalb einer SicAri-Infrastruktur realisiert werden. Das Security Token beinhaltet alle Informationen, die zum Aufbau eines Sicherheitskontextes auf der Plattform benötigt werden und ermöglicht somit die Autorisation von Benutzeraktionen und die Durchsetzung der Sicherheitsrichtlinien.

Die XPath-gesteuerte Nachrichtenverarbeitung in den Axis-Handler AMH und STH erlaubt auch die Verarbeitung von SOAP-Nachrichten, die durch andere WS-Frameworks erstellt wurden. Das beschriebene „Prefix“-Problem konnte durch die Einführung von Platzhaltern gelöst werden. Die durch die Nutzung von Web Services gewonnene Plattformunabhängigkeit konnte damit gewahrt werden.

Die Evaluationsergebnisse haben verdeutlicht, dass die Gesamtlaufzeit für die globale Authentifikation akzeptabel ist. Durch die richtige Wahl der Sicherheitsoptionen kann die Gesamtlaufzeit auf ca. 3,5 Sekunden beschränkt werden. Der SSO führt zu geringen Verzögerungen (< 600ms) und beeinflusst damit das Antwortverhalten der Plattform kaum.

Die Vorgehensweise in dieser Arbeit unterscheidet sich von der üblichen Vorgehensweise für die Entwicklung sicherer Systeme, denn in der Regel wird eine Bedrohungsanalyse vor der Implementierung vorgenommen. Dadurch können schon während der Entwicklungsphase die identifizierten Bedrohungen beachtet werden und so entsprechende Abwehrmechanismen implementiert werden.

Hier wurde die Bedrohungsanalyse allerdings erst nach der Implementierung durchgeführt, denn für die Erstellung einer Bedrohungsanalyse sind umfangreiche und detaillierte Kenntnisse über

den Aufbau und die Interna der zu untersuchenden Architektur erforderlich. Diese konnten erst, durch die intensive Beschäftigung mit der vorhandenen Implementierung und die Integration eigener Entwicklungen, erworben werden. Aus diesem Grund wurde entschieden, die Bedrohungsanalyse erst nach der Implementierung durchzuführen.

Durch diese Vorgehensweise konnten Bereiche der Plattform identifiziert werden, die vor Angriffen besonders zu schützen sind und deren gegenwärtige Implementierung angepasst werden muss. Außerdem wurde so auch festgestellt, dass die Sicherheitsoption „Verschlüsselung eines Security Token“ nur geringe Sicherheitsvorteile ergibt und in der Praxis wahrscheinlich nicht zum Einsatz kommt.

Während der Erstellung der Arbeit hat sich herausgestellt, dass die im Grundlagenkapitel umfangreich beschriebenen Kritierienkataloge für die Bedrohungsanalyse nicht verwendet werden können. Anhand der Kriterienkataloge können lediglich die Sicherheitseigenschaften der Plattform bewertet werden. Da diese von Beginn der Entwicklung maßgebend den Aufbau und das Design der Plattform beeinflusst haben und damit eines der herausragenden Merkmale der Plattform darstellen, ist es jedoch sehr zweckmäßig, die Sicherheitsfunktionen und die Qualität der Umsetzung dieser Funktionen von einer unabhängigen Instanz bewerten zu lassen. Hierfür bietet sich eine Evaluierung anhand der Common Criteria an. Dabei wird geprüft, ob die Bedrohungen, die Einsatzumgebung, die Sicherheitsziele und Sicherheitsanforderungen zusammenpassen. Außerdem werden die einzelnen Systemfunktionalitäten, und die Qualität mit der diese Funktionen umgesetzt werden, bewertet. Durch eine anschließende Zertifizierung kann die Qualität und die Vertrauenswürdigkeit der Plattform maßgeblich erhöht werden. Eine Bewertung anhand der Common Criteria ist eine sehr umfangreiche Aufgabe und konnte natürlich im Rahmen dieser Arbeit nicht umgesetzt werden.

Die WS*-Spezifikationen, wie WS-Trust, WS-Policy oder WS-SecureConversation, konnten bedauerlicherweise nicht im praktischen Einsatz untersucht werden. Hierfür waren keine Implementierungen vorhanden, die innerhalb der SicAri-Plattform hätten eingesetzt werden können. Die theoretischen Betrachtungen haben jedoch gezeigt, welche nützlichen Erweiterungen diese Spezifikationen schaffen.

Die Untersuchung und praktische Validierung der Möglichkeiten der Java ME hat gezeigt, dass sich mobile Endgeräte, ohne Erweiterungen an dem bestehenden Authentifikations- und Kommunikationskonzept vornehmen zu müssen, an die Plattform anbinden lassen. In Abhängigkeit der Konfiguration CLDC und CDC sowie der eingesetzten optionalen APIs, kann sich ein Benutzer über ein mobiles Endgerät mittels des Benutzername/Passwort- und Soft Token-basierenden Verfahrens authentifizieren. Auch der SSO kann mittels eines ASN.1-Security Token realisiert werden. Um die Vertraulichkeit und Integrität der ausgetauschten Nachrichten zu gewährleisten, kann das SSL/TLS-Protokoll verwendet werden. Mit der CDC kann sogar die Protokolloption der Client-Authentifikation umgesetzt werden.

Dies war teilweise überraschend, da ein mobiles Endgerät, wie ein Mobiltelefon oder ein PDA, nur über sehr eingeschränkte Hardwareressourcen verfügt. Mit dem weiteren Fortschritt der Technologie sollte auch die Verarbeitung von SAML-Security Tokens und die Nutzung von WS-Security-Mechanismen auf einem mobilen Endgerät möglich sein.

7.2 Ausblick

Mittlerweile existiert Axis in der Version 2. Die Entwickler dieser Version versprechen viele Neuerungen und Verbesserungen. Mit Axis 2 soll sich die Leistungsfähigkeit der Nachrichten-

verarbeitung durch das so genannte *Axis Object Model (AXIOM)*-Modell[1] weiter steigern lassen. AXIOM stellt ein Objektmodell dar, das auf Basis von StaX die Verarbeitung von SOAP-Nachrichten vereinfachen und beschleunigen soll. Der Wechsel auf Axis 2 innerhalb von SicAri kann daher evtl. die Leistungsfähigkeit der Authentifikation und des SSO verbessern.

Eine weitere wichtige Neuerung von Axis 2 ist die umfangreiche Unterstützung vieler der neuen WS*-Spezifikationen. Für SicAri bedeutet dies, dass durch diese Spezifikationen aktuelle Implementierungen ersetzt werden können und so zu mehr Flexibilität beigetragen werden kann. Die Spezifikationen, die für SicAri Vorteile bringen könnten, sind WS-Trust und WS-SecureConversation sowie WS-Policy, WS-SecurityPolicy und WS-PolicyAttachment.

Durch die WS-Trust-Spezifikation könnte der Authentifikationsmanager der SicAri-Infrastruktur zu einem Security Token Service (STS) erweitert werden. Dadurch wird das Ausstellen von Security Tokens über eine standardisierte Schnittstelle möglich. Außerdem können über den STS Security Tokens erneuert, überprüft und entwertet werden. Durch den Einsatz der WS-Trust-Spezifikation würde SicAri, im Hinblick auf die Erweiterbarkeit und Integrierbarkeit, weitere Vorteile gewinnen.

Durch den Einsatz der WS-SecureConversation-Spezifikation kann die sichere Kommunikation zwischen zwei Plattformen auf Basis von WS-Security im Hinblick auf die Leistungsfähigkeit verbessert werden. Werden mehrere Nachrichten zwischen zwei Plattformen ausgetauscht, wie beispielsweise bei der Benutzerauthentifikation über den Authentifikationsmanager, so muss für jede Verbindung der Schlüssel für die Chiffrierung der Nachricht generiert werden. Durch den Einsatz der WS-SecureConversation-Spezifikation wird zu Beginn der Kommunikation ein Sicherheitskontext aufgebaut. Dieser erlaubt es, die für die Verschlüsselung und Signatur von Nachrichten benötigten Schlüssel und Parameter für die gesamte folgende Kommunikation festzulegen.

Mit der WS-Policy-Spezifikation in Kombination mit den Spezifikationen WS-SecurityPolicy und WS-PolicyAttachment ist es möglich, die momentan in den Konfigurationsdateien der Axis-Handler STH und AMH definierten Sicherheitsoptionen, auf die Ebene einzelner Web Services auszuweiten. Durch die Erweiterung der WSDL-Beschreibung können für einen Web Service detaillierte Sicherheitsanforderungen definiert werden. So kann festgelegt werden, welcher ST-Typ ein Web Service erwartet oder durch welche Mechanismen die Nachrichtensicherung erfolgen soll. Die Leistungsfähigkeitstests haben gezeigt, dass der Einsatz der WS-Security-Mechanismen teilweise zu erheblich längeren Aufrufzeiten führt, als wenn das SSL/TLS-Protokoll eingesetzt wird. Mit WS-Policy kann nun festgelegt werden, für welchen Web Service das SSL/TLS-Protokoll ausreichend ist oder für welchen WS-Security-Mechanismen angewendet werden müssen. Die Definition der Sicherheitsanforderungen im Rahmen einer WSDL-Beschreibung, im Gegensatz zu der WS-übergreifenden Definition in einer Konfigurationsdatei, kann damit die Flexibilität verbessern.

Für Endgeräte, die keine eigene Instanz einer SicAri-Plattform ausführen, wäre die Bereitstellung einer Bibliothek, die den Zugriff auf die Plattform vereinfacht, sinnvoll. Diese könnte die Adresse des Authentifikationsmanagers und verschiedene vordefinierte Einstellungen für die Authentifikation beinhalten. Für die Java ME könnte eine solche Bibliothek auch eine, auf die Authentifikationsmechanismen der SicAri-Plattform abgestimmte, Implementierung von JAAS-Modulen beinhalten.

Außerdem wäre es sinnvoll, eine Möglichkeit bereitzustellen, mit der eine Übersicht über die verschiedenen Web Services, die in der SicAri-Infrastruktur existieren, abgerufen werden kann. Dadurch kann der Benutzer die gewünschten Dienste einer Plattform schnell finden und nutzen.

[1] `http://ws.apache.org/commons/axiom/index.html`

Auch einen Client für den Zugriff auf die Plattform über die SicAri-Remote-Shell könnte eine solche Bibliothek enthalten.
Es wäre sicherlich auch noch interessant gewesen, zu untersuchen, wie der Web Service-Zugriff auf die SicAri-Plattform unter Verwendung von anderen WS-Frameworks möglich ist. Die Plattform-, Protokoll- und Programmiersprachenunabhängigkeit ist zwar durch die Stützung auf XML prinzipiell vorhanden, doch ob alle WS-Frameworks untereinander kompatibel sind, muss untersucht werden. Die Interoperabilität von Web Services wird von der Organisation WS-I gefördert. Unterstützt ein Framework das von der WS-I definierte *Basic Profile*, so sollte der Zugriff auf die Plattform problemlos möglich sein, da die eingesetzte Web Service-Engine Axis kompatibel zum Basic Profile ist.
Denkbar wäre deshalb auch der Zugriff auf SicAri über ein mobiles Endgerät unter Verwendung von anderen Umgebungen, wie das Palm OS oder Microsoft Windows CE. Hier wären entsprechende Clients, beispielsweise in der Programmiersprache C oder unter der Plattform .NET, möglich.

Anhang A

Akronyme

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
API	Application Programming Interface
AM	Authentifikationsmanager
AMH	AuthenticationManagerHandler
AMS	Application Management Software
Ant	Another Neat Tool
ASN.1	Abstract Syntax Notation One
AXIOM	Axis Object Model
BPEL4WS	Business Process Execution Language for Web Services
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certification Authority
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CC	Common Criteria
CEM	Common Methodology for Information Security Evaluation
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CR-Verf.	Challenge Response-Verfahren
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DES	Data Encryption Standard

DOM	Document Object Model
DoS	Denial of Service
DNS	Domain Name System
DTD	Document Typ Definition
ebXML	Electronic Business XML
EAL	Evaluation Assurance Level
EVG	Evaluierungsgegenstand
FAU	Functional Requirements Security Audit
FCO	Functional Requirements Communication
FCS	Functional Requirements Cryptographic Support
FDP	Functional Requirements User Data Protection
FIA	Functional Requirements Identification and Authentication
FMT	Functional Requirements Security Management
FP	Foundation Profile
FPR	Functional Requirements Privacy
FPT	Functional Requirements Protection of the TSF
FRU	Functional Requirements Resource Utilisation
FTA	Functional Requirements Target of Evaluation Access
FTP	Functional Requirements Trusted Path/Channels
	File Transfer Protocol
GCF	Generic Connection Framework
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IDEA	International Data Encryption Algorithm
IDS	Intrusion Detection System
ID-FF	Liberty Identity Federation Framework
ID-SIS	Liberty Identity Services Interface Specifications
ID-WSF	Liberty Identity Web Services Framework
ISO	Internationalen Standardisierungsorganisation
IP	Internet Protocol
ITK, ITSK	IT-Kriterien, IT-Sicherheitskriterien

ITSEC	Information Technology Security Evaluation Criteria
ITU	International Telecommunication Union
Java EE	Java Enterprise Edition
Java ME	Java Micro Edition
Java SE	Java Standard Edition
JAAS	Java Authentication and Authorization Service
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JCRMI	Java Card Remote Method Invocation
JDK	Java Development Kit
JSSE	Java Secure Socket Extension
JSR	Java Specification Request
JVM	Java Virtual Machine
KDC	Key Distribution Center
KVM	Kilobyte Virtual Machine
LDAP	Lightweight Directory Access Protocol
MD	Message Digest
MAC	Message Authentication Code
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MITM	Man-In-the-Middle
OMG	Object Management Group
OSI	Open Systems Interconnection
PAM	Pluggable Authentication Module
PAOS	Reverse SOAP
PBP	Personal Basis Profile
PDA	Personal Digital Assistant
PDP	Policy Decision Point
PEP	Policiy Enforcement Point
PGP	Pretty Good Privacy
PIN	Persönliche Identifikationsnummer

PKI	Public Key-Infrastruktur
PP	Protection Profile
	Personal Profile
PUK	PIN Unblocking Key
QoS	Quality of Service
RA	Registration Authority
RBAC	Role-based access control
REL	Rights Expression Language
REST	Representational State Transfer
RMI	Remote Methode Invocation
RPC	Remote Procedure Calls
RSA	Rivest Shamir Adleman
RST	Request Security Token
RSTR	Request Security Token Response
SAAJ	SOAP with Attachments API for Java
SATSA	Security and Trust Services APIs
SAX	Simple API for XML
SAML	Security Assertion Markup Language
SCT	Security Context Token
SGML	Standard Generalized Markup Language
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
SSH	Secure Shell
SSO	Single Sign-On
ST	Security Target
	Security Token
STH	SecurityTokenHandler
STR	SecurityTokenReference
STRDT	STR Dereference Transform

STS	Security Token Service
SWA	SOAP with Attachments
TCB	Trusted Computing Base
TCSEC	Trusted Computer System Evaluation Critera
TGT	Ticket Granting Ticket
TLS	Transport Layer Security
TOE	Target of Evaluation
TSF	Target of Evaluation Security Functionality
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
WS	Web Services
WS-BPEL	Web Services Business Process Execution Language
WS-I	Web Services Interoperability Organization
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
WSS	Web Services Security
XACML	Extensible Access Control Markup Language
XKMS	XML Key Management Specification
XML	eXtensible Markup Language
XMLP	eXtensible Markup Language Protocol
XPath	XML Path Language
XSD	XML-Schema Definition
XSL	eXtensible Stylesheet Langugage
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformations
X-KISS	XML Key Information Service Specification
X-KRSS	XML Key Registration Service Specification
ZIS	Zentralstelle für Sicherheit in der Informationstechnik

Anhang B

WS-Frameworks

Die folgenden Tabellen (vgl. [1]) geben einen Überblick über die einzelnen Eigenschaften der existierenden WS-Frameworks. Neben den generellen Eigenschaften wird auch aufgezeigt, welche WS*-Spezifikation von den einzelnen WS-Frameworks unterstützt werden.

Feature	Axis 1.x	Axis 2.x	Celtix	JBossWS	XFire 1.2	Glassfish	OrcacleAS10g
Open Source	X	X	X	X	X	X	X
RPC-Encoding[1]	X			X		X[2]	X
REST Support[3]		X				X	X
IDEA Plugins[4]		X		X	X	X	Ant[5]
Eclipse Plugins[6]		X		X	X	X	Ant
NetBeans Plugins[7]						X	Ant
JDeveloper[8]							X
Hot Deployment[9]		X		X	?	X	X
SOAP 1.1	X	X	X	X	X	X	X
SOAP 1.2	X	X		X	X	X	X
StAX based		X		X	X	X	

Tabelle B.1: Generelle Features der existierenden WS-Frameworks

[1] Das Konzept der *Remote Procedure Calls (RPC)* bezeichnet einen Mechanismus der verteilten Programmierung, bei dem entfernte Funktionen so aufgerufen werden können, als stünden sie lokal zur Verfügung.

[2] Nur verfügbar durch das JAX-RPC 1.1 API

[3] *Representational State Transfer(REST)* [23]

[4] IDEA ist eine Entwicklungsumgebung für Java, `http://www.jetbrains.com/idea/`

[5] *Another Neat Tool (Ant)* ist ein in Java implementiertes Werkzeug zur Erstellung von Programmen und Bibliotheken aus bestehendem Quelltext

[6] Eclipse ist eine Open-Source Entwicklungsumgebung für fast alle beliebigen Programmiersprachen, `http://www.eclipse.org/`

[7] NetBeans ist vorrangig eine Entwicklungsumgebung für Java, bietet jedoch auch Unterstützung für C, C++ und Fortran, `http://www.netbeans.org/`

[8] JDeveloper ist eine Entwicklungsumgebung für Java, `http://www.oracle.com/de`

[9] Bezeichnet einen Mechanismus, der es erlaubt Java-Klassen zur Laufzeit auszutauschen bzw. neue Klassen einzuspielen und damit neue Funktionen zur Verfügung zu stellen.

Feature	Axis 1.x	Axis 2.x	Celtix	JBossWS	XFire 1.2	Glassfish	OrcacleAS10g
JAX-RPC[10]	X			X		X	X
JAX-WS[11]		X	X		X	X[12]	
JSR 181[13]	X	X	X	X	X	X	X
SAAJ (1.2/1.3)[14]	X	X		X	X	X	X
JSR 109[15]						X	X

Tabelle B.2: Unterstützte JSR der existierenden WS-Frameworks

Feature	Axis 1.x	Axis 2.x	Celtix	JBossWS	XFire 1.2	Glassfish	OrcacleAS10g
WS-Security	X	X		X	X	X	X
WS-Policy		X				X	X
WS-Trust		X				X	
WS-Secure Conversation		X				X	
WS-Security Policy		X				X	
WSDL 1.1 Support	X	X	X	X	X	X	X
WSDL 2.0 Support		X					
WS-Addressing	X	X	X	X	X	X	mit BPEL
WS-Metadata Exchange		X [16]				X	
WS-Reliable-Messaging	X	X	X			X	
WS-Eventing		X		X			
WS-Transfer		X				X	

Tabelle B.3: Unterstützte WS*-Standards der existierenden WS-Frameworks

[10] Java-API zur Implementierung von RPC auf XML-Basis

[11] Java-API zur Erstellung von Web Services

[12] JAX-RPC 1.1, JAX-WS 2.0, JAX-WS 2.1 sind in Glassfish kombiniert, außerdem ist JAX-WS 2.0 auch über die Java Standard Edition 6.0 verfügbar.

[13] Definiert eine Meta-Sprache für Web Services, die die Entwicklung von Enterprise Web Services vereinfachen und beschleunigen soll.

[14] Die *SOAP with Attachments API for Java (SAAJ)* ermöglicht es SOAP-Nachrichten und deren Anhänge zu erstellen und zu verarbeiten.

[15] Java-API zur Implementierung von Enterprise Web Services

[16] Über das Axis Erweiterungsmodul WSO2 Mex: `http://wso2.org/projects/commons/mex`

Anhang C

Security Token-Struktur

C.1 ASN1-Security Token

C.1.1 ASN1SecurityToken

```
ASN1SecurityToken ::= SEQUENCE
{
  plainSecurityToken ::= PlainSecurityToken
}
```

C.1.1.1 PlainSecurityToken

```
PlainSecurityToken ::= SEQUENCE
{
  version       ::= ASN1IA5String,
  tokenID       ::= ASN1IA5String,
  creationTime  ::= ASN1UTCTime,
  validFrom     ::= ASN1UTCTime,
  validUntil    ::= ASN1UTCTime,
  attributes    ::= ASN1SequenceOf
                    attribute ::= Attribute
}

Attribute ::= SEQUENCE
{
  key           ::= ASN1IA5String,
  value         ::= ASN1IA5String
}
```

C.1.1.2 SignedData

```
SignedData ::= SEQUENCE
{
  version Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  contentInfo ContentInfo,
  certificates
```

```
    [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
  crls
    [1] IMPLICIT CertificateRevocationLists OPTIONAL,
  signerInfos SignerInfos
}

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

C.1.1.3 SignerInfo

```
SignerInfo ::= SEQUENCE
{
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes [0] IMPLICIT Attributes OPTIONAL,
  digestEncryptionAlgorithm DigestEncryptionAlgorithmIdentifier,
  encryptedDigest EncryptedDigest,
  unauthenticatedAttributes [1] IMPLICIT Attributes OPTIONAL
}

EncryptedDigest ::= OCTET STRING

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

DigestEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

IssuerAndSerialNumber ::= SEQUENCE
{
  issuer Name,
  serialNumber CertificateSerialNumber
}

CertificateSerialNumber ::= INTEGER

Attributes ::= SET OF Attribute -- from X.501
```

C.1.1.4 AlgorithmIdentifier

```
AlgorithmIdentifier  ::= SEQUENCE
{
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
```

C.1.2 ASN1EncryptedSecurityToken

```
EncryptedASN1SecurityToken ::= SEQUENCE
{
```

```
  securityToken ::= ASN1SecurityToken
}
```

C.1.2.1 EnvelopedData

```
EnvelopedData ::= SEQUENCE
{
  version Version,
  recipientInfos RecipientInfos,
  encryptedContentInfo EncryptedContentInfo
}

RecipientInfos ::= SET OF RecipientInfo
```

C.1.2.2 RecipientInfo

```
RecipientInfo ::= SEQUENCE
{
  version Version, -- 0 for version 1.5 of PKCS#7
  issuerAndSerialNumber IssuerAndSerialNumber,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey
}

EncryptedKey ::= OCTET STRING

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

IssuerAndSerialNumber ::= SEQUENCE {
  issuer Name,
  serialNumber CertificateSerialNumber
}

CertificateSerialNumber ::= INTEGER
```

C.1.2.3 EncryptedContentInfo

```
EncryptedContentInfo ::= SEQUENCE {
  contentType ContentType,
  contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
  encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL
}

EncryptedContent ::= OCTET STRING

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
```

C.1.2.4 AlgorithmIdentifier

```
AlgorithmIdentifier  ::= SEQUENCE
```

```
{
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
```

C.2 SAMLSecurityToken

Der Aufbau des SAML-Security Token wird durch die eine SAML-Assertion festgelegt. Die SAML-Assertion Struktur ergibt sich aus der XML-Schema-Definition die hier auszugsweise aufgeführt ist.

C.2.1 SAML-Assertion

```
<schema
    targetNamespace="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified"
    version="1.1">
  <element name="Assertion" type="saml:AssertionType"/>
  <complexType name="AssertionType">
    <sequence>
      <element ref="saml:Conditions" minOccurs="0"/>
      <element ref="saml:Advice" minOccurs="0"/>
      <choice maxOccurs="unbounded">
        <element ref="saml:Statement"/>
        <element ref="saml:SubjectStatement"/>
        <element ref="saml:AuthenticationStatement"/>
        <element ref="saml:AuthorizationDecisionStatement"/>
        <element ref="saml:AttributeStatement"/>
      </choice>
      <element ref="ds:Signature" minOccurs="0"/>
    </sequence>
    <attribute name="MajorVersion" type="integer" use="required"/>
    <attribute name="MinorVersion" type="integer" use="required"/>
    <attribute name="AssertionID" type="ID" use="required"/>
    <attribute name="Issuer" type="string" use="required"/>
    <attribute name="IssueInstant" type="dateTime" use="required"/>
  </complexType>
  <element name="Conditions" type="saml:ConditionsType"/>
  <complexType name="ConditionsType">
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="saml:AudienceRestrictionCondition"/>
      <element ref="saml:DoNotCacheCondition"/>
      <element ref="saml:Condition"/>
    </choice>
    <attribute name="NotBefore" type="dateTime" use="optional"/>
    <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
  </complexType>
  <element
      name="AttributeStatement"
```

```
    type="saml:AttributeStatementType"/>
<complexType name="AttributeStatementType">
  <complexContent>
    <extension base="saml:SubjectStatementAbstractType">
      <sequence>
        <element ref="saml:Attribute" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element
    name="SubjectStatement"
    type="saml:SubjectStatementAbstractType"/>
<complexType name="SubjectStatementAbstractType" abstract="true">
  <complexContent>
    <extension base="saml:StatementAbstractType">
      <sequence>
        <element ref="saml:Subject"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="Subject" type="saml:SubjectType"/>
<complexType name="SubjectType">
  <choice>
    <sequence>
      <element ref="saml:NameIdentifier"/>
      <element ref="saml:SubjectConfirmation" minOccurs="0"/>
    </sequence>
    <element ref="saml:SubjectConfirmation"/>
  </choice>
</complexType>
<element
    name="NameIdentifier"
    type="saml:NameIdentifierType"/>
<complexType name="NameIdentifierType">
  <simpleContent>
    <extension base="string">
      <attribute name="NameQualifier"
          type="string" use="optional"/>
      <attribute name="Format" type="anyURI" use="optional"/>
    </extension>
  </simpleContent>
</complexType>
<element name="Attribute" type="saml:AttributeType"/>
<complexType name="AttributeType">
  <complexContent>
    <extension base="saml:AttributeDesignatorType">
      <sequence>
        <element
            ref="saml:AttributeValue"
            maxOccurs="unbounded"/>
      </sequence>
    </extension>
```

```
    </complexContent>
  </complexType>
  <element name="AttributeValue" type="anyType"/>
  ...
</schema>
```

C.2.2 SAML-Assertion-Beispiel

```
<Assertion
    xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
    AssertionID="sicari1"
    IssueInstant="2007-05-03T07:46:11.343Z"
    Issuer="Authentication Authority"
    MajorVersion="1"
    MinorVersion="1">
  <Conditions
    NotBefore="2007-05-03T07:46:11.343Z"
    NotOnOrAfter="2007-05-03T08:46:11.343Z">
  </Conditions>
  <AttributeStatement>
    <Subject>
      <NameIdentifier
          Format="e-mail"
          NameQualifier="https://igd.example.org/sicari">
        user@example.org
      </NameIdentifier>
    </Subject>
    <Attribute
        AttributeName="activated.roles"
        AttributeNamespace="namespace">
      <AttributeValue>
        &lt;all&gt;
      </AttributeValue>
    </Attribute>
    <Attribute
        AttributeName="authentication.method"
        AttributeNamespace="namespace">
      <AttributeValue>
        soft token
      </AttributeValue>
    </Attribute>
  </AttributeStatement>
</Assertion>
```

Anhang D

Nachrichtenmitschnitte

D.1 SOAP-Nachrichten mit SecurityToken

Die folgenden Kapitel zeigen SOAP-Nachrichten mit einem eingebetteten Security Token. Die Beispiele zeigen jeweils Nachrichten mit ASN.1- und SAML-basierenden Security Token in verschlüsselter und unverschlüsselter Form.

D.1.1 SOAP-Nachricht mit ASN1SecurityToken

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
  <wsse:BinarySecurityToken
     EncodingType="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-soap-message-security-1.0#Base64Binary"
     ValueType="ASN1SecurityToken"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-secext-1.0.xsd">
    MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMA
    M44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwl
    RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNB
    ...
  </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticateResponse soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <authenticateReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.1.2 SOAP-Nachricht mit ASN1EncryptedSecurityToken

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1EncryptedSecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIFcAIBADGCAQgwggEEAgEAMG0wZTELMAkGA1UEBhMCREUxDzANBgNVBAgTBkhlc3NlbjE
      SMBAGA1UEBxMJRGFybXN0YWR0MRAwDgYDVQQKEwdGaEctSUdEMQswCQYDVQQLEwJBODESMB
      AGA1UEAxMJU2ljQXJpLUNBAgQTcCp9MA0GCSqGSIb3DQEBAQUABIGAn1twcsPAk9fKglN7R
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticateResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <authenticateReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.1.3 SOAP-Nachricht mit SAMLSecurityToken

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security
       soapenv:actor="Assertion"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <saml:Assertion
         AssertionID="sicari2"
         IssueInstant="2007-06-11T14:29:55.265Z"
         Issuer="Authentication Authority"
         MajorVersion="1"
         MinorVersion="1"
         xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
         xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
         xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
        <saml:Conditions
           NotBefore="2007-06-11T14:29:55.265Z"
           NotOnOrAfter="2007-06-11T15:29:55.265Z"/>
        <saml:AttributeStatement>
          <saml:Subject>
            <saml:NameIdentifier
               Format="e-mail"
               NameQualifier="https://igd.example.org/sicari">
              user@example.org
            </saml:NameIdentifier>
          </saml:Subject>
```

```
  <saml:Attribute
     AttributeName="activated.roles"
     AttributeNamespace="namespace">
    <saml:AttributeValue>
      &lt;all&gt;
    </saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute
     AttributeName="authentication.method"
     AttributeNamespace="namespace">
    <saml:AttributeValue>
      security token
    </saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute
     AttributeName="authentication.platform"
     AttributeNamespace="namespace">
    <saml:AttributeValue>
      ping://89.50.240.67:50001
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm
       ="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#sicari2">
      <ds:Transforms>
        <ds:Transform
           Algorithm="http://www.w3.org/2000/09/
              xmldsig#enveloped-signature"/>
        <ds:Transform Algorithm=
           "http://www.w3.org/2001/10/xml-exc-c14n#">
          <ec:InclusiveNamespaces PrefixList=
             "code ds kind rw saml samlp typens #default"
             xmlns:ec=
             "http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod
         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>
        l2qj74dXF3QdVaK8rffwslX/PqA=
      </ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    knzBBcM0A38XflBHwzLSPRSg3xsw7tvKTTpHC27IL+RuW/jqqI9yciAd+c
    zs3CEcXQZeRMApB30DM6IoiV8p/UuuXBvB9wbHuJiKF24Aoa6IYUouUeCk
    4qHQVORf+e5wVomEI1bIyj
    oXUJfoGxgTOr05bUjUY=
  </ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>
        MIICFDCCAdGgAwIBAgIEE27wFjALBgcqhkjOOAQDBQAwZTELMAkGA1
        EUxDzANBgNVBAgTBkhlc3NlbjESMBAGA1UEBxMJRGFybXN0YWR0MRA
```

```
                QKEwdGaEctSUdEMQswCQYDVQQLEwJBODESMBAGA1UEAxMJU2ljQXJp
                ...
              </ds:X509Certificate>
            </ds:X509Data>
          </ds:KeyInfo>
        </ds:Signature>
      </saml:Assertion>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticateResponse soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <authenticateReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.1.4 SOAP-Nachricht mit chiffriertem SAMLSecurityToken

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security
       soapenv:actor="Encryption"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <xenc:EncryptedKey Id="EncKeyId-12068486">
        <xenc:EncryptionMethod
           Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <ds:X509Data>
              <ds:X509IssuerSerial>
                <ds:X509IssuerName>
                   CN=SicAri-CA,OU=A8,O=FhG-IGD,L=Darmstadt,ST=Hessen,C=DE
                </ds:X509IssuerName>
                <ds:X509SerialNumber>
                   326118013
                </ds:X509SerialNumber>
              </ds:X509IssuerSerial>
            </ds:X509Data>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
            Q/lJRBIFZaqSDMl8Cjl0tIdqqr+uZNCW7g64q5o0mZQh+pUuiNTBJTTDPh8HrLtuH
            FQNpBuhYztxX+rke+WzlZSSMjTnWv+ZhYp0jxk0IJw+Fd9+gDxjj4gRF66vIKRhxC
            WQi8KAJagfvA0rZ12eQv0fQtgsUevIcu0sBAc=
          </xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
```

```
          <xenc:DataReference URI="#EncDataId-28591556"/>
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
    </wsse:Security>
    <wsse:Security
       soapenv:actor="Assertion"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <xenc:EncryptedData
         Id="EncDataId-28591556"
         Type="http://www.w3.org/2001/04/xmlenc#Element">
        <xenc:EncryptionMethod
           Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
        <xenc:CipherData>
          <xenc:CipherValue>
            D+Ph48Bxx6SxO64Uksm+IkufJk537M3GwlNF9QKSeDywsMPuTHtGhSSd3i
            T5rKZfmxDpFSS8UagXYp/WYbRxOQYTidKQ7pFjoLumsZ0sCJa7b/Hhsg5/
            oq+nXoXtyPuaO+elCu9AB7WJ+CnhhnMXzWf+mTv/IUN2o4DHt8I56YiRnv
            ...
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedData>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticateResponse soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <authenticateReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope
```

D.2 Aufruf eines Web Service mit ST und WSS-Mechanismen

Am Beispiel des Aufrufs des SicAri-`Hello`-WS, werden in den folgenden Kapiteln, die einzelnen implementierten WS-Security-Mechanismen dargestellt. Der `Hello`-WS ist eine erweiterte Implementierung des bekannten `HelloWorld`-Programmierbeispiels. Die Beispiele zeigen jeweils den Aufruf der `getByte()`-Methode, als erstes ohne WS-Security-Mechanismen, als zweites mit Verschlüsselung des SOAP-Body, als drittes mit SOAP-Body-Signatur und schließlich mit Signatur und Verschlüsselung. Für die Authentifikation bei der Plattform, auf der der Hello-WS publiziert ist, enthält jede Nachricht ein ASN.1-kodiertes ST.

D.2.1 Hello-WS mit ASN.1-ST

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
```

```
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsGByq
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwlEYX
      Jtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNBc
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:getByte
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://hello.webservice.sicari.de.test"/>
  </soapenv:Body>
</soapenv:Envelope>
```

D.2.2 Hello-WS mit ASN.1-ST & SOAP-Body Verschlüsselung

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security
       soapenv:actor="Encryption"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <xenc:EncryptedKey Id="EncKeyId-8763735">
        <xenc:EncryptionMethod
           Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <ds:X509Data>
              <ds:X509IssuerSerial>
                <ds:X509IssuerName>
                  CN=SicAri-CA,OU=A8,O=FhG-IGD,L=Darmstadt,ST=Hessen,C=DE
                </ds:X509IssuerName>
                <ds:X509SerialNumber>
                  326038323
                </ds:X509SerialNumber>
              </ds:X509IssuerSerial>
            </ds:X509Data>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
            NfqNPSOKxzbVNENiuNQG//uWxT8fRpQ9RoPIiHG6poPXPXfUhYvxW18A+SdYYX0qv
            pjg7yzFBqPvog4e8kUb7scj4Rr/byUJoLMxWVo09oKE8ew1rIfgKFZstLxA2oWGKo
            BFx4VIcq7mJ4jdd3aq/u6GoeHJM501I0Wc6FnIhdU=
          </xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#EncDataId-21705033"/>
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
    </wsse:Security>
    <wsse:BinarySecurityToken
```

```
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsGByq
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwlEYX
      Jtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNBc
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <xenc:EncryptedData
       Id="EncDataId-21705033"
       Type="http://www.w3.org/2001/04/xmlenc#Content">
      <xenc:EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
      <xenc:CipherData>
        <xenc:CipherValue>
          F7INtw+M/5+fXkVDstbHidBuB92pnsmf/Qc/gqmnYqOr5uYSMc5S2neDP37r
          EEulET2vLVgoMeSO+QKUSPMN/m7A+g+PtllICSvdWHW77L6wwyjJt4jJy48W
          Hg6P3icXrKryrWOb84DhZk8hwtRkywJP+jnjmLdIYYQ4VWhGwtvgyUpXoWk7
          ...
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soapenv:Body>
</soapenv:Envelope>
```

D.2.3 Hello-WS mit ASN.1-ST & SOAP-Body Signatur

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security
       soapenv:actor="Signature"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <ds:Signature
         Id="Signature-3992148"
         xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#id-28480896">
            <ds:Transforms>
              <ds:Transform
                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>
              +EoS2Ie7pPtseiErMxiSy1ibCBc=
            </ds:DigestValue>
          </ds:Reference>
```

```
        </ds:SignedInfo>
        <ds:SignatureValue>
          cOCO5w0uz8/i3VoNi0FK1T1JK/A6E3/l0LfOGoTncbRP9FxmskOZyGk3/A9v
          DVk+lmKA469GS9k7pqyg1TX4Plrd9fG/PcPrICic1Rdp1+xrzA8Y0wexG33n
          Gntl+vL2tqv3x7/C0ZVmASi1Bo42QQKeiaeUU=
        </ds:SignatureValue>
        <ds:KeyInfo Id="KeyId-5745446">
          <wsse:SecurityTokenReference
             wsu:Id="STRId-27613516"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <ds:X509Data>
              <ds:X509IssuerSerial>
                <ds:X509IssuerName>
                  CN=SicAri-CA,OU=A8,O=FhG-IGD,
                  L=Darmstadt,ST=Hessen,C=DE
                </ds:X509IssuerName>
                <ds:X509SerialNumber>
                  326117309
                </ds:X509SerialNumber>
              </ds:X509IssuerSerial>
            </ds:X509Data>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsGByq
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwlEYX
      Jtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNBc
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body
     wsu:Id="id-28480896"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <ns1:getByte
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://hello.webservice.sicari.de.test"/>
  </soapenv:Body>
</soapenv:Envelope>
```

D.2.4 Hello-WS mit ASN.1-ST & SOAP-Body Verschlüsselung und Signatur

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security
       soapenv:actor="Encryption"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
```

```
        oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <xenc:EncryptedKey Id="EncKeyId-10294373">
      <xenc:EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <wsse:SecurityTokenReference>
          <ds:X509Data>
            <ds:X509IssuerSerial>
              <ds:X509IssuerName>
                CN=SicAri-CA,OU=A8,O=FhG-IGD,L=Darmstadt,ST=Hessen,C=DE
              </ds:X509IssuerName>
              <ds:X509SerialNumber>
                326038323
              </ds:X509SerialNumber>
            </ds:X509IssuerSerial>
          </ds:X509Data>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>
          W/eFarOkXFPmmQUVUe0yQfk3S97DNq+2KdEING9mLTJK2em0qOJtacnOPV
          IybKJeMZRs1exX5vZh3WMYwUI5bfewk7LxX7UIcyex62EWVBkg61GA5XFt
          9fCW+0czx+ZJFxPmLXy8/up4zs5Scq9kZXezinM4=
        </xenc:CipherValue>
      </xenc:CipherData>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#EncDataId-13177791"/>
      </xenc:ReferenceList>
    </xenc:EncryptedKey>
  </wsse:Security>
  <wsse:Security
     soapenv:actor="Signature"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <ds:Signature
       Id="Signature-15954614"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <ds:SignatureMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#id-13177791">
          <ds:Transforms>
            <ds:Transform
               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:Transforms>
          <ds:DigestMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>
            +/lT211npOh3nEAHDYCfNYZkIYA=
          </ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>
        V32eDZzQ+YXYskMfW3+PySaTswh9iwNzIhkRs8UuPZq0BF6TKO5/ZKUdMXup
        4nyZYbpivwozPlvZkCqPRLGosu8ZAfpxnRRRLeDwDoakhBPMDchnjNuZ8XDA
        qSBQHwbQv9SkYoVy0EhYWZU9BD2qphfjUKjbPTnhJYE=
      </ds:SignatureValue>
```

```
        <ds:KeyInfo Id="KeyId-15732298">
          <wsse:SecurityTokenReference
             wsu:Id="STRId-3820003"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <ds:X509Data>
              <ds:X509IssuerSerial>
                <ds:X509IssuerName>
                  CN=SicAri-CA,OU=A8,O=FhG-IGD,
                  L=Darmstadt,ST=Hessen,C=DE
                </ds:X509IssuerName>
                <ds:X509SerialNumber>
                  326117309
                </ds:X509SerialNumber>
              </ds:X509IssuerSerial>
            </ds:X509Data>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsG
      ByqGSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQH
      EwlEYXJtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQD
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body
     wsu:Id="id-13177791"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <xenc:EncryptedData
      Id="EncDataId-13177791"
      Type="http://www.w3.org/2001/04/xmlenc#Content">
      <xenc:EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
      <xenc:CipherData>
        <xenc:CipherValue>
          FzktWop1PferIjB5IRR6VVf1y1RXKo9nxI0zigmbr+TXgwm9ya2KhUKwNCDHLpLw
          UZ933oXlM6xmEyDJAv6C1tGjerdnsMVFI8ur/f4OVEgCF0eyEtSRv3oxbzCAiYa8
          OxEVrCniZCWyXFz3DosAeiwU4rYRX3AB/8w5h7XB6XTqLsxfW2eE47H+wjZqUcgC
          ...
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3 SOAP-Nachrichten der WS-Methoden des Authentifikationsmanagers

Im Folgenden werden, jeweils beispielhaft, eine Anfrage- und eine Antwortnachrichten der WS-Methoden des Authentifikationsmanagers dargestellt.

D.3.1 getAuthenticationMethods(String userID)

D.3.1.1 Anfrage

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <ns1:getAuthenticationMethods
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://authentication.sicari.de">
     <userID
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="soapenc:string">
       sicari2
     </userID>
   </ns1:getAuthenticationMethods>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.1.2 Antwort

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getAuthenticationMethodsResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <getAuthenticationMethodsReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:base64Binary">
        9wsl5nvExh+MyWaDLhW8NavQLfrCmXLrdvNbNYK6xqZ3lQ==
      </getAuthenticationMethodsReturn>
    </ns1:getAuthenticationMethodsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.2 authenticate(byte m, String userID, byte[] req)

D.3.2.1 Anfrage

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:authenticate soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/"
```

```
       xmlns:ns1="http://authentication.sicari.de">
      <authMethod href="#id0"/>
      <userID
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </userID>
      <authRequest
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:base64Binary">
        VDI2U349w/pyraXvKngjBdbMEcvuYli6XHCL1esIHpTr27B0eEhHWXlfDNZ+xG
        Q2g7OvJPFyQg2HjzZQAzlCrYKFyDDUVUQNnkwuC4BFvLT8dWn6xOpgjuC5XBa6
        w+8Bc2csbKaWb/rzKtHeaKZeYo+/oXmA0XG1M=
      </authRequest>
    </ns1:authenticate>
    <multiRef id="id0" soapenc:root="0" soapenv:encodingStyle
       ="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="xsd:byte">
      -14
    </multiRef>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.2.2 Antwort

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsGByq
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwlEYX
      Jtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNBc
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticateResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <authenticateReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.3 cancel(String userID)

D.3.3.1 Anfrage

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:cancel soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <userID
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </userID>
    </ns1:cancel>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.3.2 Antwort

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:cancelResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de"/>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.4 authenticate(String tokenId)

D.3.4.1 Anfrage

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAW
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQ
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticate soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <tokenId
```

```
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </tokenId>
    </ns1:authenticate>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.4.2 Antwort

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsGByq
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwlEYX
      Jtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNBc
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
     <ns1:authenticateResponse
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="http://authentication.sicari.de">
       <authenticateReturn
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
          xsi:type="soapenc:string">
         sicari2
       </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.5 authenticate(String tokenId, String tokenType)

D.3.5.1 Anfrage

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:BinarySecurityToken
       EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
       ValueType="ASN1SecurityToken"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      MIIEMwIBATELMAkGBSsOAwIaBQAwAwYBAKCCAhgwggIUMIIB0aADAgECAgQTbvAWMAsGByq
      GSM44BAMFADBlMQswCQYDVQQGEwJERTEPMA0GA1UECBMGSGVzc2VuMRIwEAYDVQQHEwlEYX
      Jtc3RhZHQxEDAOBgNVBAoTB0ZoRy1JR0QxCzAJBgNVBAsTAkE4MRIwEAYDVQQDEwlTaWNBc
```

```
      ...
    </wsse:BinarySecurityToken>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:authenticate
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <tokenId
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </tokenId>
      <tokenType
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        saml
      </tokenType>
    </ns1:authenticate>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.5.2 Antwort

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security
       soapenv:actor="Assertion"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <saml:Assertion
         AssertionID="sicari2"
         IssueInstant="2007-06-11T16:24:30.343Z"
         Issuer="Authentication Authority"
         MajorVersion="1"
         MinorVersion="1"
         xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
         xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
         xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
        <saml:Conditions
           NotBefore="2007-06-11T16:24:30.343Z"
           NotOnOrAfter="2007-06-11T17:24:30.343Z"/>
        <saml:AttributeStatement>
          <saml:Subject>
            <saml:NameIdentifier
               Format="e-mail"
               NameQualifier="https://igd.example.org/sicari">
              user@example.org
            </saml:NameIdentifier>
          </saml:Subject>
          <saml:Attribute
             AttributeName="authentication.method"
             AttributeNamespace="namespace">
            <saml:AttributeValue>
              security token
            </saml:AttributeValue>
          </saml:Attribute>
```

```
          <saml:Attribute
             AttributeName="activated.roles"
             AttributeNamespace="namespace">
            <saml:AttributeValue>
              &lt;all&gt;
            </saml:AttributeValue>
          </saml:Attribute>
          <saml:Attribute
             AttributeName="authentication.platform"
             AttributeNamespace="namespace">
            <saml:AttributeValue>
              ping://89.50.240.67:50001
            </saml:AttributeValue>
          </saml:Attribute>
        </saml:AttributeStatement>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:SignedInfo>
            <ds:CanonicalizationMethod
               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#sicari2">
              <ds:Transforms>
                <ds:Transform
                   Algorithm="http://www.w3.org/2000/09/
                      xmldsig#enveloped-signature"/>
                <ds:Transform
                   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                  <ec:InclusiveNamespaces
                     PrefixList="code ds kind rw saml samlp typens #default"
                     xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                </ds:Transform>
              </ds:Transforms>
              <ds:DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>
                AELNWGE6W7TTcYc96Z+ayiesleQ=
              </ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
          <ds:SignatureValue>
            jqhjQXfn5PgFTqdR0tBGTjAXcgLTW7lwT/SIJpqOdp4Y8wXunjLQaAZRke8Dmh0rV
            i9OG6SgUPDvHH9+3CFZ1YskOyW8FF4MGIVx8lMDYj3tceWs9gX9X/WDYLsiSQYMfA
            2Yh3DK977ZOSnVoQwTg1SFSaA7vuy/k28KoXbKXIQ=
          </ds:SignatureValue>
          <ds:KeyInfo>
            <ds:X509Data>
              <ds:X509Certificate>
                MIICFDCCAdGgAwIBAgIEE27wFjALBgcqhkjOOAQDBQAwZTELMAkGA1UEBhMCR
                EUxDzANBgNVBAgTBkhlc3NlbjESMBAGA1UEBxMJRGFybXN0YWR0MRAwDgYDVQ
                QKEwdGaEctSUdEMQswCQYDVQQLEwJBODESMBAGA1UEAxMJU2ljQXJpLUNBMB4
                ...
              </ds:X509Certificate>
            </ds:X509Data>
          </ds:KeyInfo>
        </ds:Signature>
      </saml:Assertion>
    </wsse:Security>
  </soapenv:Header>
```

```
  <soapenv:Body>
    <ns1:authenticateResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://authentication.sicari.de">
      <authenticateReturn
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string">
        sicari2
      </authenticateReturn>
    </ns1:authenticateResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

D.3.6 SOAP-Fault

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>
        soapenv:Server.userException
      </faultcode>
      <faultstring>
        de.sicari.authentication.AuthenticationException:
        Authentication failed: password does not match
      </faultstring>
      <detail>
        <ns1:hostname
           xmlns:ns1="http://xml.apache.org/axis/">
           pberthol
        </ns1:hostname>
      </detail>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

Anhang E

Konfigurationsdateien und -skripte

E.1 Neue Konfigurationsdateien

E.1.1 authentication.conf

```
###
#
# This is a sample configuration file for the Authentication service.
#
# The following parameters are recognized:
#
#   login.profile    = <default-login-profile>
#   token.type       = asn1 | saml | xml
#   token.validity   = <default-token-validity-in-minutes>
#  [callback.handler = <login-callback-handle-class>]
#
###
#
# Author:  Jan Peters
# Version: $Id$
#
###

login.profile    = default

token.type       = saml
token.validity   = 60

callback.handler = com.sun.security.auth.callback.TextCallbackHandler
#callback.handler = com.sun.security.auth.callback.DialogCallbackHandler
```

E.1.2 ws-st-handler.conf

```
###
#
# This is a sample configuration file for the Axis-SecurityTokenHandler
#
# The following parameters are recognized:
#
#   authentication.level = 0 | 1 | 2 | 3
#     3 : SSL with Client-Authentication and SecurityToken
#     2 : SSL and SecurityToken
#     1 : SecurityToken
#     0 : None
#
#   The authentication levels 2 and 3 will only be treated by the server side
#   of the corresponding handler. Thus, there is no ssl session validation on the
#   client side.
```

```
#   The authentication levels 1 to 3 imply, if the access to the underlying
#   service should be granted the message must include a valid ST of
#   the requesting user.
#
#   token.encryption = 0 | 1
#     If enabled the SecurityTokenHandler encrypts the SecurityToken
#
# The parameters {request|response}.message.{encryption|signature} determine if
# a message on the sender side should be encrypted resp. a signature should be
# calculated. On the recipient side these parameters will not be treated. Thus,
# if a message contains a signature the signature will be validated. If a message
# is encrypted it will be decrypted, if possible.
#
#   request.message.encryption = 0 | 1
#     If enabled the SecurityTokenHandler encrypts the Request-SOAP-Body
#
#   request.message.signature = 0 | 1
#     If enabled the SecurityTokenHandler signs the Request-SOAP-Body
#
#   response.message.encryption = 0 | 1
#     If enabled the SecurityTokenHandler encrypts the Response-SOAP-Body
#
#     Attention: response message encryption is only possible if also
#                request message signature is enabled or the authentication
#                level 3 (ssl with client authentication) is selected
#                -> based on the signature the DName of the sender respectively
#                   the peer principal DName is reused to catch the certificate
#                   for message encryption.
#
#   response.message.signature = 0 | 1
#     If enabled the SecurityTokenHandler signs the Response-SOAP-Body
#
#   xpath.filter.[#] = A list of XPath-Expression that will be used to
#                      filter all messages that should not be handled
#                      by the SecurityTokenHandler
#
###
#
# Author:  Peter Berthold
# Version: $Id$
#
###

authentication.level = 2

token.encryption            = 0
request.message.encryption  = 0
request.message.signature   = 0
response.message.encryption = 0
response.message.signature  = 0

xpath.filter.1 = //*[starts-with(namespace-uri(),'urn:uddi-org:api')]
xpath.filter.2 = //*[namespace-uri() = 'http://authentication.sicari.de']
xpath.filter.3 = //*[namespace-uri() = 'http://xml.apache.org/axis/wsdd/']
```

E.1.3 ws-am-handler.conf

```
###
#
# This is a sample configuration file for the Axis-AuthenticationManagerHandler
#
# The following parameters are recognized:
#
#   authentication.level = 0 | 1 | 2 | 3
#     3 : SSL with Client-Authentication and SecurityToken
#     2 : SSL and SecurityToken
#     1 : SecurityToken
```

```
#     0 : None
#
#   The authentication levels 2 and 3 will only be treated by the server side
#   of the corresponding handler. Thus, there is no ssl session validation on the
#   client side.
#   The authentication levels 1 to 3 imply, that a ST will be integrated to the
#   exchanged SOAP message if this is needed by means of the authentication
#   protocol.
#
#   token.encryption = 0 | 1
#     If enabled the AuthenticationManagerHandler encrypts the SecurityToken
#
#     Attention: A SecurityToken can be included in a message to the
#                authenticaton manager (when a authenticate request is made,
#                based on a security token created by the local platform admin)
#                as well as in a message form the authentication manager (when
#                the user could be successfully authenticated)
#                In the first case we can determine the instance for which the
#                token encryption should be done via the ping service. In the
#                second case this is not possible, in this case we must
#                determine this instance via the certificate of a enabled
#                message signature respectively via ssl with client
#                authentication
#
# The parameters {request|response}.message.{encryption|signature} determine if
# a message on the sender side should be encrypted resp. a signature should be
# calculated. On the recipient side these parameters will not be treated. Thus,
# if a message contains a signature the signature will be validated. If a message
# is encrypted it will be decrypted, if possible.
#
#   request.message.encryption = 0 | 1
#     If enabled the AuthenticationManagerHandler encrypts the Request-SOAP-Body
#
#   request.message.signature = 0 | 1
#     If enabled the AuthenticationManagerHandler signs the Request-SOAP-Body
#
#   response.message.encryption = 0 | 1
#     If enabled the AuthenticationManagerHandler encrypts the Response-SOAP-Body
#
#     Attention: response message encryption is only possible if also
#                request message signature is enabled or the authentication
#                level 3 (ssl with client authentication) is selected
#                -> based on the signature the DName of the sender respectively
#                   the peer principal DName is reused to catch the certificate
#                   for message encryption.
#
#   response.message.signature = 0 | 1
#     If enabled the AuthenticationManagerHandler signs the Response-SOAP-Body
#
#   xpath.filter.[#] = A list of XPath-Expression that will be used to
#                      filter all messages that should not be handled
#                      by the AuthenticationManagerHandler
#
###
#
# Author:  Peter Berthold
# Version: $Id$
#
###

authentication.level = 2

token.encryption           = 0
request.message.encryption  = 0
request.message.signature   = 0
response.message.encryption = 0
response.message.signature  = 0
```

```
xpath.filter.1 = not(//*[namespace-uri() = 'http://authentication.sicari.de'])
# => all messages that do not conform to the specified namespace will be skipped
#    by this handler, thus all messages that conform will be processed
```

E.2 Erweiterte Konfigurationsdateien und -skripte

E.2.1 login.conf

```
/*
 * This is a sample login configuration file, which configures a cascade of
 * LoginModules to be traversed during the SicAri login process.
 *
 * Each module may accept different kinds of options which should be
 * found in the documentation of the corresponding module class.
 *
 * A detailed documentation about the syntax of this file
 * can also be found under
 * http://java.sun.com/j2se/1.5.0/docs/api/javax/security/auth/login/Configuration.html
 *
 * NOTICE: JAAS provides its own variable substitution which uses the same
 *         syntax as the variable substitution of SicAri. In order to bypass
 *         the JAAS mechanism to be substituted by the SicAri, variables
 *         intended to be subject of substitution must meet the following
 *         syntax:
 *                   $_{<variable-name>}
 */

/*
 * Author:  Matthias Pressfreund
 * Version: $Id$
 */


default
{
    de.sicari.kernel.security.GlobalPKCS11LoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.pkcs11.conf";
    de.sicari.kernel.security.GlobalJKSLoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.jks.conf";
    de.sicari.kernel.security.GlobalPasswordLoginModule SUFFICIENT;
    de.sicari.kernel.security.PKCS11LoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.pkcs11.conf";
    de.sicari.kernel.security.JKSLoginModule SUFFICIENT
                    keymaster.conf="$_{sicari.etc}/keymaster.jks.conf";
    de.sicari.kernel.security.PasswordLoginModule REQUISITE;
};

remote
{
    de.sicari.kernel.security.SSLLoginModule SUFFICIENT;
    de.sicari.kernel.security.PasswordLoginModule REQUISITE;
};

webservice
{
    de.sicari.kernel.security.SecurityTokenLoginModule REQUISITE;
};

other
{
    de.sicari.kernel.security.PasswordLoginModule REQUISITE;
};
```

E.2.2 rc.admin

```
##############################################################################
#
# Setup the the currently authenticated user as authentication manager,
# thus the user to sign security tokens.
# (Can only be called once in the name of a valid user - Subject)
#
Publish -key ${WhatIs:AUTHENTICATION} \; -setAuthenticationManager

##############################################################################
#
# Setup the the currently authenticated user as platform authenticator,
# thus the user which identifies SSL client/server side.
# (Can only be called once in the name of a valid user - Subject)
#
Publish -key ${WhatIs:SSL_MASTER} \; -setPlatformIdentity

# Check if SSL-Master has successfully been initialized
if eval ${status} eq "-1"; then
  setenv SSL disable
  source ${sicari.etc}/rc.dependencies
fi

##############################################################################
#
# Setup network services
#
source ${sicari.etc}/rc.network

##############################################################################
#
# Setup the the currently authenticated user as platform identity,
# who signs and decrypte SOAP messages.
# (Can only be called once in the name of a valid user - Subject)
#
${WSE_STH} java de.sicari.authentication.SecurityTokenHandler
  -setPlatformIdentity
${WSE_AMH} java de.sicari.authentication.AuthenticationManagerHandler
  -setPlatformIdentity

##############################################################################
#
# Hide the introduction window
#
${SICARI_INTRO} java de.sicari.kernel.gui.SicariSplashScreen -hide

##############################################################################
#
# Setup local user services
#
source ${sicari.etc}/rc.local

##############################################################################
#
# Logout PF-Administrator and thereby request user login
#
echo
echo "User login..."
echo "-------------"

exit

##############################################################################
```

E.2.3 rc.conf

```
#
# This file contains options that can be enabled or disabled.
# Furthermore it contains some default settings.
# It is loaded before server specific configuration scripts.
# Hence, its defines are global unless overloaded by subsequent
# scripts.
#
# Port address settings should be overwritten within
# the specific configuration scripts ("*.conf").
#
# Author  : Jan Peters
# Revision: $Revision: 1.6 $
# Date    : $Date: 2005/10/28 14:36:05 $
#


# Introduction settings
# ---------------------
#
setenv SICARI_INTRO   enable


# Network settings
# ----------------
#
# NOTE: enable STANDALONE if the SicAri middleware
#       should start without connection to the
#       global SicAri directory server.
#
setenv STANDALONE   disable


# SSL settings
# ------------
#
setenv SSL   enable


# Ping server shell settings
# --------------------------
#
setenv PING        enable
setenv PING_PORT   50000


# Remote shell settings
# ---------------------
#
setenv RSHD        enable
setenv RSHD_PORT   48000
setenv SSHD        enable
setenv SSHD_PORT   48010


# HTTP daemon settings
# --------------------
#
#   HTTPS_HOST_VERIFICATION: for HttpsURLConnection
#
#     HTTPS_NO_HOST_VERIFICATION
#     -> The peer hostname of a HTTPS URL connection is not verified.
#     HTTPS_SIMPLE_HOST_VERIFICATION
#     -> If the certificate of the SSL peer is given, the CN value of
#        the subject DN has to equal the peer's hostname.
#     HTTPS_STRICT_HOST_VERIFICATION
#     -> The certificate of the SSL peer has to be given, and the
```

```
#        value of the subject DN has to equal the peer's hostname.
#
#   HTTPS_TRUST_MANAGEMENT
#
#     HTTPS_NO_TRUST_CHECK
#     -> No trust check is done at all.
#     HTTPS_ISSUER_TRUST_CHECK
#     -> The peer's certificate chain is verified and the root CA
#        certificate is compared against the trusted certificates
#        of the KeyMaster.
#
setenv HTTPS_NO_HOST_VERIFICATION       0
setenv HTTPS_SIMPLE_HOST_VERIFICATION   1
setenv HTTPS_STRICT_HOST_VERIFICATION   2

setenv HTTPS_NO_TRUST_CHECK             0
setenv HTTPS_ISSUER_TRUST_CHECK         1

setenv MIMETYPES     ${sicari.etc}/mime.types
setenv HTTPD         enable
setenv HTTPD_PORT    8080
setenv HTTPSD        enable
setenv HTTPSD_PORT   8880
setenv WEB           enable

setenv HTTPS_HOST_VERIFICATION   ${HTTPS_NO_HOST_VERIFICATION}
setenv HTTPS_TRUST_MANAGEMENT    ${HTTPS_NO_TRUST_CHECK}


# WebService engine settings
# --------------------------
#
setenv WSE                       enable
setenv WSE_CORE                  enable
setenv WSE_CORES                 enable
setenv WSE_BASE_URL              webservice
setenv WSE_BASE_DIR              ${sicari.base}/lib/axis/default
setenv WSE_UDDI                  enable
setenv WSE_UDDI_HTTP_CONF_FILE   ${sicari.etc}/uddi.http.conf
setenv WSE_UDDI_HTTPS_CONF_FILE  ${sicari.etc}/uddi.https.conf
setenv WSE_UDDI_CONF_FILE        ${WSE_UDDI_HTTPS_CONF_FILE}
setenv WSE_MANAGER               enable
setenv WSE_MANAGER_CONF_FILE     ${sicari.etc}/webservice.conf
setenv WSE_CLEAN                 enable


# WebService security handler settings
# ------------------------------------
#
setenv WSE_STH             enable
setenv WSE_STH_CONF_FILE   ${sicari.etc}/ws-st-handler.conf


# AuthenticationManager settings
# ------------------------------
#
setenv AUTH_MANAGER        disable
setenv WSE_AMH             enable
setenv WSE_AMH_CONF_FILE   ${sicari.etc}/ws-am-handler.conf


# AuthenticationService settings
# ------------------------------
#
setenv AUTHENTICATION_CONF   ${sicari.etc}/authentication.conf


# Security settings
```

```
# -----------------
#
setenv TRUSTED_CERTS_STORE   file:///${sicari.etc}/trusted-ca.jks
setenv KEYMASTER_CONF        file:///${sicari.etc}/keymaster.jks.conf


# IdentityManager settings
# ------------------------
#
setenv LDAP   enable

setenv IDENTITY_LDAP_CONF    ${sicari.etc}/identity.ldap.conf
setenv IDENTITY_LDAPS_CONF   ${sicari.etc}/identity.ldaps.conf
setenv IDENTITY_LOCAL_CONF   ${sicari.etc}/identity.local.conf

setenv IDENTITY_CONF   ${IDENTITY_LDAPS_CONF}


# PolicyService settings
# ------------------------------
#
setenv POLICY_CONF   ${sicari.etc}/policy.conf


# Envision settings
# -----------------
#
setenv ENVISION             enable
setenv ENVISION_PLUGINS     disable
setenv ENVISION_CONF_FILE   ${sicari.etc}/envision.conf


# Application settings
# --------------------
#
setenv CEBIT              disable
setenv CONTEXT_RESOURCE   file:///${sicari.etc}/checklist-db.xml

setenv MOCKUP        disable
setenv MOCKUP_DATA   ${sicari.etc}/mockup
```

E.2.4 rc.dependencies

```
#
# This file adjusts option settings according to
# module dependencies.
# It is loaded after global and user settins have
# been parsed and before server specific configuration
# scripts.
#
# Author  : Jan Peters
# Revision: $Revision: 1.6 $
# Date    : $Date: 2006-06-26 07:54:10 +0200 (Mon, 26 Jun 2006) $
#

echo
echo "Check service dependencies"
echo "--------------------------"


#
# If network has been configured as STANDALONE:
#
# - Disable LDAP requests to SicAri directory server.
# - Disable UDDI requests to SicAri directory server
#
```

```
if eval ${STANDALONE} eq enable; then
  if eval ${LDAP} eq enable; then
    echo "STANDALONE configuration -> LDAP disabled."
    setenv LDAP disable
  fi
  if eval ${WSE_UDDI} eq enable; then
    echo "STANDALONE configuration -> WSE_UDDI disabled."
    setenv WSE_UDDI disable
  fi
fi


#
# Configure modules according to LDAP setting.
#
if eval ${LDAP} eq disable; then
  if eval ${IDENTITY_CONF} eq ${IDENTITY_LDAP_CONF}; then
    echo "LDAP is not available -> Use ID-Manager configuration
      "${IDENTITY_LOCAL_CONF}"."
    setenv IDENTITY_CONF ${IDENTITY_LOCAL_CONF}
  fi
  if eval ${IDENTITY_CONF} eq ${IDENTITY_LDAPS_CONF}; then
    echo "LDAP is not available -> Use ID-Manager configuration
      "${IDENTITY_LOCAL_CONF}"."
    setenv IDENTITY_CONF ${IDENTITY_LOCAL_CONF}
  fi
fi


#
# Disable all SSL dependent modules,
# if SSL has been deactivated.
#
if eval ${SSL} eq disable; then
  if eval ${PING} eq enable; then
    echo "SSL is not available -> PING disabled."
    setenv PING disable
  fi
  if eval ${SSHD} eq enable; then
    echo "SSL is not available -> SSHD disabled."
    setenv SSHD disable
  fi
  if eval ${HTTPSD} eq enable; then
    echo "SSL is not available -> HTTPSD disabled."
    setenv HTTPSD disable
  fi
fi


#
# Configure modules according to SSL setting.
#
if eval ${SSL} eq disable; then
  if eval ${WSE_UDDI_CONF_FILE} eq ${WSE_UDDI_HTTPS_CONF_FILE}; then
    echo "SSL is not available -> Use UDDI configuration
      "${WSE_UDDI_HTTP_CONF_FILE}"."
    setenv WSE_UDDI_CONF_FILE ${WSE_UDDI_HTTP_CONF_FILE}
  fi
fi
if eval ${SSL} eq disable; then
  if eval ${IDENTITY_CONF} eq ${IDENTITY_LDAPS_CONF}; then
    echo "SSL is not available -> Use IdentityManager configuration
      "${IDENTITY_LDAP_CONF}"."
    setenv IDENTITY_CONF ${IDENTITY_LDAP_CONF}
  fi
fi
```

```
#
# Disable WEB servlets, if neither HTTP
# nor HTTPS daemon has been enabled.
#
if eval ${HTTPD} eq disable; then
  if eval ${HTTPSD} eq disable; then
    if eval ${WEB} eq enable; then
      echo "Neither HTTPD nor HTTPSD available -> WEB disabled."
      setenv WEB disable
    fi
  fi
fi


#
# Check dependencies for the Web Service Engine
#
if eval ${WSE} ne enable; then
  if eval ${WSE_CORE} eq enable; then
    echo "WSE is not available -> WSE_CORE disabled."
    setenv WSE_CORE disable
  fi
  if eval ${WSE_CORES} eq enable; then
    echo "WSE is not available -> WSE_CORES disabled."
    setenv WSE_CORES disable
  fi
  if eval ${WSE_UDDI} eq enable; then
    echo "WSE is not available -> WSE_UDDI disabled."
    setenv WSE_UDDI disable
  fi
  if eval ${WSE_MANAGER} eq enable; then
    echo "WSE is not available -> WSE_MANAGER disabled."
    setenv WSE_MANAGER disable
  fi
  if eval ${WSE_STH} eq enable; then
    echo "WSE is not available -> WSE_STH disabled."
    setenv WSE_STH disable
  fi
  if eval ${WSE_AMH} eq enable; then
    echo "WSE is not available -> WSE_AMH disabled."
    setenv WSE_AMH disable
  fi
fi
if eval ${HTTPD} eq disable; then
  if eval ${WSE_CORE} eq enable; then
    echo "HTTPD is not available -> WSE_CORE disabled."
    setenv WSE_CORE disable
  fi
fi
if eval ${HTTPSD} eq disable; then
  if eval ${WSE_CORES} eq enable; then
    echo "HTTPSD is not available -> WSE_CORES disabled."
    setenv WSE_CORES disable
  fi
fi
if eval ${WSE_CORE} eq disable; then
  if eval ${WSE_CORES} eq disable; then
    if eval ${WSE_UDDI} eq enable; then
      echo "Neither WSE_CORE nor WSE_CORES is available -> WSE_UDDI disabled."
      setenv WSE_UDDI disable
    fi
    if eval ${WSE_MANAGER} eq enable; then
      echo "Neither WSE_CORE nor WSE_CORES is available -> WSE_MANAGER disabled."
      setenv WSE_MANAGER disable
    fi
  fi
fi
if eval ${WSE_UDDI} eq disable; then
```

```
  if eval ${WSE_MANAGER} eq enable; then
    echo "WSE_UDDI is not available -> WSE_MANAGER disabled."
    setenv WSE_MANAGER disable
  fi
fi


#
# Set corresponding option variable
# to clean AXIS base directory from
# previous configuration files.
#
if eval ${WSE_CLEAN} eq enable; then
  setenv CLEAN_AXIS_CONFIG "-clean"
else
  setenv CLEAN_AXIS_CONFIG ""
fi


#
# Disable ENVISION_PLUGINS, if
# the the envision framework is not available.
#
if eval ${ENVISION} eq disable; then
  if eval ${ENVISION_PLUGINS} eq enable; then
    echo "ENVISION is not available -> ENVISION_PLUGINS disabled."
    setenv ENVISION_PLUGINS disable
  fi
fi


#
# Set corresponding option variable
# to publish servlets within the HTTP
# server.
#
if eval ${HTTPD} eq enable; then
  setenv HTTP "-http"
else
  setenv HTTP ""
fi


#
# Set corresponding option variable
# to publish servlets within the HTTPS
# server.
#
if eval ${HTTPSD} eq enable; then
  setenv HTTPS "-https"
else
  setenv HTTPS ""
fi


#
# If SicAri intro splash screen has been enabled,
# set the appropriate initialization option to
# display the splash screen.
#
# Otherwise, just set the option to patch the
# AWT/Swing bug.
#
if eval ${SICARI_INTRO} eq enable; then
  setenv INIT_SICARI_INTRO "-show"
else
  setenv INIT_SICARI_INTRO "-patch"
fi
```

E.2.5 rc.main

```
################################################################################
#
# Original Version
#
# Boots a SicAri kernel based on a given configuration
# script. The boot scripts assumes that the following
# properties are defined when starting the VM:
#
# -Djava.ext.dirs=${SICARI_BASE}/lib\
# -Djava.security.policy=${SICARI_ETC}/java.policy\
# -Dde.fhg.igd.logging.config=${SICARI_ETC}/sicari.logging\
# -Dsicari.base=${SICARI_BASE}
# -Dsicari.etc=${SICARI_ETC}
# -Dsicari.log=${SICARI_LOG}
#
# where
#
# ${SICARI_BASE} is
#   The main sicari directory, should be ${USER}/sicari
#
# ${SICARI_ETC} is
#   The directory where all the configuration files
#   and boot scripts reside, should be ${USER}/sicari/etc
#   or ${USER}/sicari/cvs/etc (for developers)
#
# ${SICARI_LOG} is
#   The directory where all the log files should be
#   written to.
#
# A number of other scripts will be read and run by
# this one. The 'rc.conf' script allows to switch
# some features of the server on and off without too
# much thinking. Have a look at the sample provided
# with this script. Please remember that slashes are
# used as file separators even though the current
# platform's path separator is different (e.g. a '\'
# as on a windoze box).
#
# If you change anything herein then replace 'Original'
# at the beginning of this script by something else
# such as 'Custom'.
#
# Author  : Jan Peters
# Revision: $Revision: 1.8 $
# Date    : $Date: 2005/10/21 11:19:00 $
#
################################################################################


################################################################################
#
# Setup rudimentary server and shell support.
# The ConsoleFilter handles input and output stream
# redirection among threads. It is the heart of the
# pipeline functionality, and can be used to redirect
# stdin, stdout, and stderr on a per agent basis as well.
# It is required by the shell as well.
#
echo
echo "Basic services"
echo "--------------"
echo -n "Setup:"

echo -n " WhatIs"
java de.sicari.util.WhatIs -init ${sicari.etc}/whatis.conf
```

```
echo -n " ConsoleFilter"
java de.sicari.util.ConsoleFilter

echo "."


##############################################################################
#
# Read in the custom configuration script, overrides
# unwanted global options and defines server specific
# parameters such as port numbers.
#
echo
echo "Configuration"
echo "-------------"

#
# Load global options and predefines.
#
echo "Global: "${sicari.etc}"/rc.conf"
source ${sicari.etc}/rc.conf

#
# Read in the custom configuration script, overrides
# unwanted global options and defines server specific
# parameters such as port numbers.
#
if eval ${0} ne ""; then
  echo "Custom: "${0}
  source ${0}
else
  echo "Custom: <none>"
fi


##############################################################################
#
# Dependency checks
#
source ${sicari.etc}/rc.dependencies


##############################################################################
#
# Display introduction window and thereby initialize Java AWT
#
java de.sicari.kernel.gui.SicariSplashScreen ${INIT_SICARI_INTRO}

##############################################################################
#
# Read in aliases for commands; makes life a little easier.
#
source ${sicari.etc}/rc.aliases


##############################################################################
#
# Setup cryptographic security services
#
source ${sicari.etc}/rc.crypto


##############################################################################
#
# IMPORTANT: Setup basic security services and
# show login-prompt for platform administrator.
# All subsequently loaded services are installed
# on behalf of the platform administrator, until
```

```
# logout-command at the end of this file.
#
echo
echo "Security services"
echo "-----------------"

echo -n "Core security:"
echo -n " keymaster"
java de.sicari.kernel.security.KeyMasterImpl -publish ${KEYMASTER_CONF}
  ${TRUSTED_CERTS_STORE}
echo -n " sslmaster"
java de.sicari.kernel.security.SSLMasterImpl -publish ${HTTPS_TRUST_MANAGEMENT}
  ${HTTPS_HOST_VERIFICATION}
echo "."

echo -n "Security services:"
echo -n " identity"
java de.sicari.kernel.security.IdentityManager -publish ${IDENTITY_CONF}
echo -n " policy"
java de.sicari.kernel.security.PolicyServiceImpl -publish ${POLICY_CONF}
echo -n " auth"
java de.sicari.kernel.security.AuthenticationServiceImpl -publish
  ${AUTHENTICATION_CONF}
echo "."


# Login PF-Administrator
#
#   => All subsequent services are loaded as user 'root'
#
echo
echo "Login PF-Administrator"
echo "----------------------"
su ${PF_ADMIN} -s ${sicari.etc}/rc.admin -p keystore=${SECURITY_TOKEN}

if eval ${status} eq "-1"; then
 echo
  echo
  echo "------------------------------"
  echo " PF-Administrator login failed!"
  echo " => Terminate platform."
  echo "------------------------------"
  echo
  shutdown 1
fi

################################################################################
```

E.2.6 rc.network

```
################################################################################
#
# Original Version
#
# Boots a SicAri kernel based on a given configuration
# script. The boot scripts assumes that the following
# properties are defined when starting the VM:
#
# -Djava.ext.dirs=${SICARI_BASE}/lib\
# -Djava.security.policy=${SICARI_ETC}/java.policy\
# -Dde.fhg.igd.logging.config=${SICARI_ETC}/sicari.logging\
# -Dsicari.base=${SICARI_BASE}
# -Dsicari.etc=${SICARI_ETC}
# -Dsicari.log=${SICARI_LOG}
#
# where
#
```

```
# ${SICARI_BASE} is
#   The main sicari directory, should be ${USER}/sicari
#
# ${SICARI_ETC} is
#   The directory where all the configuration files
#   and boot scripts reside, should be ${USER}/sicari/etc
#   or ${USER}/sicari/cvs/etc (for developers)
#
# ${SICARI_LOG} is
#   The directory where all the log files should be
#   written to.
#
# A number of other scripts will be read and run by
# this one. The 'rc.conf' script allows to switch
# some features of the server on and off without too
# much thinking. Have a look at the sample provided
# with this script. Please remember that slashes are
# used as file separators even though the current
# platform's path separator is different (e.g. a '\'
# as on a windoze box).
#
# If you change anything herein then replace 'Original'
# at the beginning of this script by something else
# such as 'Custom'.
#
# Author  : Jan Peters
# Revision: $Revision: 1.8 $
# Date    : $Date: 2005/10/21 11:19:00 $
#
################################################################################


################################################################################
#
# Setup rudimentary server and shell support.
# The ConsoleFilter handles input and output stream
# redirection among threads. It is the heart of the
# pipeline functionality, and can be used to redirect
# stdin, stdout, and stderr on a per agent basis as well.
# It is required by the shell as well.
#
echo
echo "Basic services"
echo "--------------"
echo -n "Setup:"

echo -n " WhatIs"
java de.sicari.util.WhatIs -init ${sicari.etc}/whatis.conf

echo -n " ConsoleFilter"
java de.sicari.util.ConsoleFilter

echo "."


################################################################################
#
# Read in the custom configuration script, overrides
# unwanted global options and defines server specific
# parameters such as port numbers.
#
echo
echo "Configuration"
echo "-------------"

#
# Load global options and predefines.
#
```

```
echo "Global: "${sicari.etc}"/rc.conf"
source ${sicari.etc}/rc.conf

#
# Read in the custom configuration script, overrides
# unwanted global options and defines server specific
# parameters such as port numbers.
#
if eval ${0} ne ""; then
  echo "Custom: "${0}
  source ${0}
else
  echo "Custom: <none>"
fi


################################################################################
#
# Dependency checks
#
source ${sicari.etc}/rc.dependencies


################################################################################
#
# Display introduction window and thereby initialize Java AWT
#
java de.sicari.kernel.gui.SicariSplashScreen ${INIT_SICARI_INTRO}

################################################################################
#
# Read in aliases for commands; makes life a little easier.
#
source ${sicari.etc}/rc.aliases


################################################################################
#
# Setup cryptographic security services
#
source ${sicari.etc}/rc.crypto


################################################################################
#
# IMPORTANT: Setup basic security services and
# show login-prompt for platform administrator.
# All subsequently loaded services are installed
# on behalf of the platform administrator, until
# logout-command at the end of this file.
#
echo
echo "Security services"
echo "-----------------"

echo -n "Core security:"
echo -n " keymaster"
java de.sicari.kernel.security.KeyMasterImpl -publish ${KEYMASTER_CONF}
  ${TRUSTED_CERTS_STORE}
echo -n " sslmaster"
java de.sicari.kernel.security.SSLMasterImpl -publish ${HTTPS_TRUST_MANAGEMENT}
  ${HTTPS_HOST_VERIFICATION}
echo "."

echo -n "Security services:"
echo -n " identity"
java de.sicari.kernel.security.IdentityManager -publish ${IDENTITY_CONF}
echo -n " policy"
```

```
java de.sicari.kernel.security.PolicyServiceImpl -publish ${POLICY_CONF}
echo -n " auth"
java de.sicari.kernel.security.AuthenticationServiceImpl -publish
  ${AUTHENTICATION_CONF}
echo "."


# Login PF-Administrator
#
#   => All subsequent services are loaded as user 'root'
#
echo
echo "Login PF-Administrator"
echo "----------------------"
su ${PF_ADMIN} -s ${sicari.etc}/rc.admin -p keystore=${SECURITY_TOKEN}

if eval ${status} eq "-1"; then
 echo
  echo
  echo "------------------------------"
  echo " PF-Administrator login failed!"
  echo " => Terminate platform."
  echo "------------------------------"
  echo
  shutdown 1
fi

###############################################################################
```

E.2.7 sicari.conf

```
#
# SicAri-Platform Configuration (1)
#
# Author  : Jan Peters
# Revision: $Revision: 1.1 $
# Date    : $Date: 2005/10/28 14:34:53 $
#

setenv AUTH_MANAGER   enable

setenv PF_ADMIN         sicari1
setenv SECURITY_TOKEN   ${sicari.etc}/token/sicari1.jks
setenv WSE_BASE_DIR     ${sicari.base}/lib/axis/sicari1

setenv PING_PORT     50001
setenv RSHD_PORT     48001
setenv SSHD_PORT     48011
setenv HTTPD_PORT    8081
setenv HTTPSD_PORT   8881
```

E.2.8 webservice.conf

```
###############################################################################
#
#  Example webservice configuration file
#
#  Author  : Matthias Pressfreund
#  Version : $Id: webservice.conf,v 1.3 2005/10/21 11:19:00 jpeters Exp $
#
###############################################################################
#
#  This file is needed to start the WebserviceManager.
#
```

```
#  It contains a set of entries, each starting with an <Environment-path>
#  element followed by a comma separated list of <interface-class-name>
#  definitions.
#
#  After the WebserviceManager has been started, the specified
#  <Environment-path> elements will be watched for objects implementing one
#  or more of the <interface-class-name> assignments.
#  Once as a matching object has been published, it will be deployed as
#  a webservice, using the corresponding Environment path as the
#  webservice name.
#  Accordingly, whenever a watched object is retracted, the corresponding
#  webservice will be undeployed.
#
#  The syntax for a single configuration entry is:
#
#    <Environment-path>='*'|<interface-class-name>[{,<interface-class-name>}]
#
#    where:
#
#      <Environment-path> may be any canonical path. Wildcards such as
#      '*' (matching any corresponding path element) and '-' (matching
#      any corresponding path element as well as any following path elements)
#      are supported.
#      Notice: Relative <Environment-path> definitions will be automatically
#      referenced to the root path ('/').
#
#      <interface-class-name> definitions must be fully qualified and
#      resolvable. Alternatively, in case all interfaces shall be accepted,
#      an asterisk ('*') may be used instead.
#
#   Furthermore, variable substitution is supported for system properties
#   (${<property-name>}) and WhatIs variables (${WhatIs:<whatis-name>}).
#
#   Only those methods specified in the given interfaces will be deployed
#   as Web service methods.
#
################################################################################

# Automatically publish the AuthenticationManager as Web service
${WhatIs:AUTH_MANAGER}=de.sicari.authentication.AuthenticationManagerWS

#${my.webservice}=my.webservice.Interface
#${WhatIs:public}/-=*
/webservice/test/*=*
```

E.2.9 whatis.conf

```
#
# The list of static global definitions in the SicAri server.
# This list must be loaded into 'WhatIs' using a call such
# as 'java de.sicari.util.WhatIs -init <url>' where
# <url> is a file URL to this file.
#
# This file defines the name space structure of important
# objects in a SicAri server. Programs depend on the integrity
# of the keys, so don't change them unless you *exactly* know
# what you are doing. The values can be changed to suit the
# needs of your server configuration. It goes without saying
# that changes of values can be fatal if a server configuration
# or classes hardcode paths instead of using WhatIs, because
# inconsistencies are preprogrammed.
#
# Author : Jan Peters
# Version: $Id: whatis.conf,v 1.7 2005/06/29 07:38:31 jpeters Exp $
#
```

```
# The class providing the current SicAri version
#
SICARI_VERSION = /public/version

# Security related constants.
#
KEYMASTER = /security/keymaster
THREADS_FILTER = /security/threads
SSL_MASTER = /security/sslmaster
CERTFINDER = /public/certfinder
AUTHENTICATION = /security/authentication
AUTH_MANAGER = /security/auth-manager
IDENTITY = /security/identity
POLICY = /security/policy

# Miscellaneous network services
#
PING = /network/ping
HTTP_SERVER = /network/httpd
HTTPS_SERVER = /network/httpsd
HTTP_RESOURCES = /network/resources
RSH_SERVER = /network/rshd
SSH_SERVER = /network/sshd

# Plugin registry for Envision viewer
#
ENVISION = /restricted/envision

# This path is used as the prefix of the URL string returned
# by de.sicari.service.AbstractService.docs()
#
SICARI_DOCS_URL = http://www.sicari.de/docs/javaapi/

# Web Service Engine
#
WEBSERVICES = /webservice/admin
UDDI        = /webservice/uddi


# Paths for miscellaneous services
#

# Mockup: Path for mockup service
MOCKUP_SERVICE = /restricted/userinfo

# CeBIT: Checklist Context Database
CONTEXT_DB = /context/checklist-db
```

Anhang F

Inhalt der CD

F.1 Übersicht

Die CD enthält eine Sammlung aller relevanten Daten, die während der Bearbeitung dieser Arbeit benutzt bzw. erstellt wurden. Im Folgenden wird der Aufbau der CD beschrieben.

F.2 Hierarchie

- **readme.txt**
  Enthält den Aufbau und den Inhalt der CD.
- **docs**
  - **abstracts**: Beinhaltet die 1-seitige Zusammenfassung dieser Arbeit in englischer und deutscher Ausführung.
  - **slides**: Die Präsentationsfolien dieser Arbeit am Fraunhofer IGD im PDF-Format.
  - **thesis**: Die LaTeX-Quellen dieser Arbeit sowie die endgültige Version dieser Arbeit im PDF-Format.
- **mobile**
  Enthält verschiedene Bibliotheken und Projekte für den Zugriff auf die SicAri-Plattform unter Verwendung eines mobilen Endgeräts mit der Java ME. Außerdem eine Installationsanleitung für Netbeans und die Add-Ons für die Entwicklung von Java ME-Anwendungen.
  - **kSOAP2**: Bibliotheken für das kSOAP2 WS-Framework
  - **netbeans**: Netbeans-Projekte für CLDC und CDC
  - **WSOAP**: Bibliotheken für das WSOAP WS-Framework
- **papers**
  Enthält Dokumente, die in dieser Arbeit referenziert wurden, im PDF-Format. Der Dateiname leitet sich aus dem jeweiligen BibTex-Schlüssel ab.
- **sicari**
  Eine lauffähige Version der SicAri-Plattform inklusive der neuen Implementierungen die durch diese Arbeit entstanden sind.

- **sources**
  - **etc**: Neue und angepasste Konfigurationsdateien.
  - **classes**: Neue und angepasste Java-Klassen.
  - **lib**: Neue Bibliotheken von Dritt-Anbietern.
- **test-protocols**
  Dieses Verzeichnis enthält die im Rahmen der Leistungsfähigkeitstests erstellten Auswertungen.

Literaturverzeichnis

[1] Stack Comparison. 2007. http://wiki.apache.org/ws/StackComparison (02.04.2007). [Seite 237]

[2] JTC 1/SC 29/WG 11. ISO/IEC 21000-5:2004: Information technology – Multimedia framework (MPEG-21) – Part 5: Rights Expression Language (1st ed.). Technical report, International Organization for Standardization (ISO), 2003. [Seite 70]

[3] Alexandre Alves and Assaf Arkin. OASIS Web Services Business Process Execution Language (WSBPEL) v2.0. Technical report, OASIS, 2006. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v20-rddl.html (14.12.2006). [Seite 29]

[4] John J. Barton, Satish Thatte, and Henrik Frystyk Nielsen. SOAP Messages with Attachments. Technical report, W3C, 2000. http://www.w3.org/TR/SOAP-attachments (18.01.2007). [Seite 70]

[5] Bruce Schneier. Attack Trees. Technical report, Dr. Dobb's Journal, 1999. http://www.schneier.com/paper-attacktrees-ddj-ft.html (11.06.2007). [Seite 184]

[6] BSI. Deutsche IT-Sicherheitskriterien (Grünbuch). Technical report, Bundesamt für Sicherheit in der Informationtechnik, 1990. http://www.bsi.bund.de/zertifiz/itkrit/itgruend.pdf (18.12.2006). [Seiten 10, 11 und 12]

[7] BSI. Common Criteria Gemeinsame Kriterien für die Prüfung und Bewertung der Sicherheit von Informationstechnik. Technical report, Bundesamt für Sicherheit in der Informationstechnik, 2003. http://www.bsi.bund.de/literat/faltbl/F06CommonCriteria.pdf (27.12.2006). [Seite 18]

[8] BSI. Schutzprofile nach Common Criteria. Technical report, Bundesamt für Sicherheit in der Informationstechnik, 2003. http://www.bsi.bund.de/literat/faltbl/F26SchutzprofileCC.pdf (27.12.2006). [Seite 20]

[9] Bärbel Burkhard and Guido Laures. SOA - Wertstiftendes Architekur-Paradigma. *OBJEKTspektrum*, 06, 06 2003. http://www.sigs.de/publications/os/2003/06/burkhard_OS_06_03.pdf (11.12.2006). [Seite 28]

[10] Larry Cable and Thorick Chow. JSR 173: Streaming API for XML. Technical report, Java Community Process, 2004. http://jcp.org/en/jsr/detail?id=173 (14.12.2006). [Seite 34]

[11] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI Spec Technical Committee. Technical report, OASIS, 2004. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm (14.12.2006). [Seiten 29 und 39]

[12] JTC 1/SC 34 Commitee. ISO 8879 Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML). Technical report, International Organization for Standardization (ISO), 1986. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387\&ICS1=35\&ICS2=240\&ICS3=30 (14.12.2006). [Seite 30]

[13] JTC1 Commitee. ISO 7498 Open Systems Interconnection – Basic Reference Model The Basic Model. Technical report, International Organization for Standardization (ISO), 1994. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip (14.12.2006). [Seite 30]

[14] Fujitsu Software Corporation, Novell Inc, Oracle Corporation, and Sun Microsystems. WS-Reliability 1.1. Technical report, OASIS, 2004. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf (22.01.2007). [Seite 88]

[15] Thomas DeMartini, Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security Rights Expression Language (REL) Token Profile 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16687/oasis-wss-rel-token-profile-1.1.pdf (18.01.2007). [Seite 70]

[16] Yuri Demchenko, Leon Gommans, Cees de Laat, and Bas Oudenaarde. Web Services and Grid Security Vulnerarbilities and Threads Analysis and Model. Technical report, University of Amsterdam, 2005. http://staff.science.uva.nl/~demch/papers/grid2005-threats-xws-grid-analysis.pdf (15.11.2006). [Seite 198]

[17] Rüdiger Dierstein. Sicherheit in der Informationstechnik–der Begriff IT-Sicherheit. *Informatik Spektrum*, 27(4):343–353, 2004. http://springerlink.metapress.com/content/13b6x34xu34u9u3h/?p=5f0158fe217e4bc9b711ee8659cc2cc1\&pi=6 (13.12.2006). [Seite 4]

[18] Rüdiger Dierstein. IT-Sicherheit und ihre Besonderheiten - Duale Sicherheit -. *Vorlesungsskript TU München*, 2006. http://www.lrr.in.tum.de/zope/lectures/courses/WS06/itsicher/files/skripte/DualSi069.pdf (13.12.2006). [Seite 5]

[19] D. Eastlake. US Secure Hash Algorithm 1 (SHA1). Technical report, Network Working Group Request for Comments, 2001. http://tools.ietf.org/html/rfc3174 (31.12.2006). [Seite 55]

[20] OASIS ebXML Messaging Services Technical Committee. ebXML Technical Specifications. Technical report, OASIS, 2001. http://www.ebxml.org/specs/index.htm (14.12.2006). [Seite 29]

[21] Claudia Eckert. *IT-Sicherheit Konzepte-Verfahren-Protokolle (4.Auflage)*. Oldenbourg, 2006. [Seiten 3, 4, 5, 6, 7, 8, 10, 14, 16, 20, 22, 24, 41, 42, 44, 45, 46, 50, 56, 106, 181, 182, 185 und 208]

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1 - RFC2616. Technical report, Network Working Group Request for Comments, 1999. http://tools.ietf.org/html/rfc2616 (14.12.2006). [Seiten 36 und 43]

[23] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (28.03.2007). [Seite 237]

[24] Network Working Group Request for Comments. File Transfer Protocol (FTP) - RFC 959. Technical report, Network Working Group Request for Comments, 1985. http://tools.ietf.org/html/rfc959 (14.12.2006). [Seite 36]

[25] Network Working Group Request for Comments. Multipurpose Internet Mail Extensions - RFC 2045. Technical report, Network Working Group Request for Comments, 1996. http://www.ietf.org/rfc/rfc2045.txt (14.12.2006). [Seite 70]

[26] Network Working Group Request for Comments. IPSec - Security Architecture for the Internet Protocol - RFC 2401. Technical report, Network Working Group Request for Comments, 1998. http://tools.ietf.org/html/rfc2401 (14.12.2006). [Seiten 55 und 68]

[27] Network Working Group Request for Comments. The TLS Protocol Version 1.0 - RFC 2246. Technical report, Network Working Group Request for Comments, 1999. http://tools.ietf.org/html/rfc2246 (17.01.2007). [Seite 67]

[28] Network Working Group Request for Comments. Simple Mail Transfer Protocol (SMTP) - RFC 2821. Technical report, Network Working Group Request for Comments, 2001. http://tools.ietf.org/html/rfc2821 (14.12.2006). [Seite 36]

[29] Network Working Group Request for Comments. Internet X.509 Public Key Infrastructure - RFC 3280. Technical report, Network Working Group Request for Comments, 2002. http://tools.ietf.org/html/rfc3280 (01.01.2007). [Seite 57]

[30] Network Working Group Request for Comments. The Kerberos Network Authentication Service (V5) - RFC 4120. Technical report, Network Working Group Request for Comments, 2005. http://tools.ietf.org/html/rfc4120 (08.01.2007). [Seiten 40 und 41]

[31] Network Working Group Request for Comments. The Base16, Base32, and Base64 Data Encodings - RFC 4648. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4648 (02.01.2007). [Seiten 58 und 59]

[32] Network Working Group Request for Comments. The Transport Layer Security (TLS) Protocol Version 1.1 - RFC 4346. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4346 (17.01.2007). [Seite 67]

[33] Apache Software Foundation. Axis Architecture Guide Version 1.2. Technical report. http://ws.apache.org/axis/java/architecture-guide.html (29.01.2007). [Seiten 107 und 110]

[34] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutzhandbuch*. BSI - Bundesamt für Sicherheit in der Informationstechnik, 2005. http://www.bsi.bund.de/gshb/deutsch/download/itgshb_2005.pdf (17.12.2006). [Seite 182]

[35] Simson Garfinkel. *PGP. Pretty Good Privacy. Encryption for Everyone: Pretty Good Privacy*. O'Reilly Media, 1994. [Seite 55]

[36] Europäische Gemeinschaft. Information Technology Security Evaluation Criteria (ITSEC). Technical report, Europäische Gemeinschaft, 1991. http://www.bsi.bund.de/zertifiz/itkrit/itsec-dt.pdf (20.12.2006). [Seiten 13 und 14]

[37] Eric Giguere. Understanding J2ME Application Models. Technical report, SUN, 2002. http://developers.sun.com/techtopics/mobility/midp/articles/models(25.04.2007). [Seiten 90 und 91]

[38] Web Services Architecture Working Group. Web Services Architecture Requirements. Technical report, Web Services Architecture Working Group, 2004. http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/ (11.12.2006). [Seite 29]

[39] Samudra Gupta. Service Oriented Architecture. *Java Boutique*, 2004. http://javaboutique.internet.com/tutorials/serv_orient/article.html (11.12.2006). [Seite 26]

[40] Sayed Hashimi. Service-Oriented Architecture Explained. *BEA Dev2Dev*, 2004. http://dev2dev.bea.com/pub/a/2004/05/soa_hashimi.html (11.12.2006). [Seite 26]

[41] Frederick Hirsch. Web Services Security SOAP Messages with Attachments (SwA) Profile 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16672/wss-v1.1-spec-os-SwAProfile.pdf (18.01.2007). [Seite 70]

[42] Jesper Holgersson and Eva Söderström. Web Service Security - Vulnerabilites and threats within the context of WS-Security. *IEEE*, Standardization and Innovation in Information Technology, 2005. The 4th Conference on:138–146, 2005. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1563803 (11.12.2006). [Seite 198]

[43] John Hughes and Scott Cantor et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS, März 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf (10.01.2007). [Seiten 46 und 47]

[44] John Hughes and Scott Cantor et al. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS, März 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf (10.01.2007). [Seite 48]

[45] John Hughes and Scott Cantor et al. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS, März 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf (10.01.2007). [Seite 48]

[46] IBM and Microsoft. Web Services Inspection Language. Technical report, 2001. http://www-128.ibm.com/developerworks/library/specification/ws-wsilspec/ (23.01.2007). [Seite 88]

[47] IBM, BEA Systems, Microsoft, and ... Web Services Secure Conversation Language (WS-SecureConversation). Technical report, 2005. http://www-128.ibm.com/developerworks/library/specification/ws-secon/ (23.01.2007). [Seite 81]

[48] IBM, BEA Systems, Microsoft, SAP AG, Computer Associates, Sun Microsystems, and webMethods. Web Services Metadata Exchange (WS-MetadataExchange) Version 1.1. Technical report, 2006. http://www-128.ibm.com/developerworks/library/specification/ws-mex (22.01.2007). [Seite 87]

[49] IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, and VeriSign. Web Services Policy Attachment (WS-PolicyAttachment) Version 1.2. Technical report, 2006. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polatt/ (21.01.2007). [Seite 84]

[50] IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, and VeriSign. Web Services Policy Framework (WS-Policy) Version 1.2. Technical report, 2006. http://www-128.ibm.com/developerworks/library/specification/ws-polfram/ (21.01.2007). [Seite 77]

[51] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business Process Execution Language for Web Services version 1.1. Technical report, 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel (23.01.2007). [Seite 88]

[52] IBM, BEA Systems, Microsoft, Computer Associates, Sun Microsystems, and TIBCO Software. Web Services Eventing (WS-Eventing). Technical report, 2004. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-eventing/ (23.01.2007). [Seite 88]

[53] IBM, BEA Systems, Microsoft, and TIBCO Software. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). Technical report, 2005. ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf (21.01.2007). [Seite 88]

[54] IBM, BEA Systems, Microsoft, VeriSign, and RSA Security. WS-Federation: Active Requestor Profile Version 1.0. Technical report, 2003. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-fedact (23.01.2007). [Seite 83]

[55] IBM, BEA Systems, Microsoft, VeriSign, and RSA Security. WS-Federation: Passive Requestor Profile Version 1.0. Technical report, 2003. http://www-128.

ibm.com/developerworks/webservices/library/specification/ws-fedpass (23.01.2007). [Seite 83]

[56] Anthony Nadalin (IBM), Marc Goodner (Microsoft), Martin Gudgin (Microsoft), Abbie Barbir (Nortel), and Hans Granqvist (VeriSign). Web Services Security Policy Language (WS-SecurityPolicy) 1.2. Technical report, OASIS, 2006. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/ws-securitypolicy-1.2-spec-cd-01.pdf (21.01.2007). [Seite 85]

[57] Microsoft IBM. Security in a Web Services World: A Proposed Architecture and Roadmap. Technical report, International Business Machine and Microsoft Corporation, 2002. ftp://www6.software.ibm.com/software/developer/library/ws-secmap.pdf (18.12.2006). [Seiten 67, 68, 80 und 84]

[58] JFranks, P. Hallam-Baker, J. Hosetler, S. Lawrence, P. Leach, A. Luotinen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication - RFC2617. Technical report, Network Working Group Request for Comments, 1999. http://tools.ietf.org/html/rfc2617 (15.05.2007). [Seite 123]

[59] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Technical report, Network Working Group Request for Comments, 1992. http://tools.ietf.org/html/rfc2104 (31.12.2006). [Seite 55]

[60] S. Lehtinen and C. Lonvick. The Secure Shell (SSH) Protocol Assigned Numbers - RFC 4250. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4250 (18.01.2007). [Seite 68]

[61] W. Scott Means. *The Book of SAX: The Simple API for XML*. No Starch Press, 2002. [Seite 32]

[62] Microsoft. Microsoft .NET Passport Technical Overview. Technical report, Microsoft Cooperation, 2001. [Seiten 43, 44 und 45]

[63] Microsoft. Microsoft .NET Passport Review Guide. Technical report, Microsoft Cooperation, 2003. http://www.microsoft.com/net/services/passport/review_guide.asp (09.01.2007). [Seiten 43 und 45]

[64] Microsoft. Bedrohungsanalyse. Technical report, Microsoft, 2004. http://www.microsoft.com/germany/msdn/library/security/ErhoehenDerSicherheitVonWebanwendungen/secmod76.mspx?mfr=true (28.10.2006). [Seite 209]

[65] Microsoft. Designing for Securability. Technical report, Microsoft Cooperation, 2007. http://msdn2.microsoft.com/en-us/library/aa291875(VS.71).aspx (12.06.2007). [Seite 184]

[66] Ronald Monzillo, Chris Kaler, Anthony Nadalin, and Phillip Hallem-Baker. Web Services Security SAML Token Profile 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf (18.01.2007). [Seite 69]

[67] A.G. Morgan. Pluggable Authentication Modules (PAM). Technical report, OpenPAM working group, 2002. http://www.kernel.org/pub/linux/libs/pam/pre/doc/current-draft.txt (17.05.2007). [Seite 120]

[68] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. Web Services Trust Language (WS-Trust) 1.3. Technical report, OASIS, 2006. http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf (22.01.2007). [Seiten 78, 79 und 80]

[69] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security Kerberos Token Profile 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf (18.01.2007). [Seite 70]

[70] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security: SOAP Message Security 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf (18.01.2007). [Seite 69]

[71] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security UsernameToken Profile 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf (18.01.2007). [Seiten 69 und 72]

[72] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security X.509 Certificate Token Profile 1.1. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf (18.01.2007). [Seite 70]

[73] Jan Oetting and Jan Peters. MW1: Interoperability with Component Standards and Web Services, Version 1.0. Technical report, SicAri Konsortium, 2004. http://www.sicari.de/fileadmin/content/MiddlewarePlattform/projektarbeiten/SicAri-MW1.pdf (07.01.2007). [Seite 117]

[74] Jan Oetting, Jan Peters, Ulrich Pinsdorf, Taufiq Rochaeli, and Ruben Wolf. PF3: Specification of the SicAri Architecture, Version 2.0. Technical report, SicAri Konsortium, 2004. http://www.sicari.de/fileadmin/content/MiddlewarePlattform/projektarbeiten/pf3-specification-architecture.pdf (05.01.2007). [Seiten 106 und 111]

[75] Jan Oetting, Taufiq Rochaeli, and Ruben Wolf. PF2: Requirements for the SicAri Architecture, Version 1.17. Technical report, SicAri Konsortium, 2004. http://www.sicari.de/fileadmin/content/MiddlewarePlattform/projektarbeiten/pf2-requirements-architecture.pdf (11.12.2006). [Seiten 95, 99 und 100]

[76] United States Government Department of Defense (DoD). Trusted Computer System Evaluation Criteria (TCSEC). Technical Report 5200.28-STD, United States Government Department of Defense (DoD), 1985. http://www.dynamoo.com/orange/fulltext.htm (18.12.2006). [Seiten 9 und 10]

[77] OMG. Common Object Request Broker Architecture (CORBA/IIOP). Technical report, Object Management Group, 2002. http://www.omg.org/technology/documents/corba_spec_catalog.htm (14.12.2006). [Seite 26]

[78] Rolf Oppliger. Microsoft .NET Passport: A Security Analysis. Technical report, IEEE, Juli 2003. http://ieeexplore.ieee.org/xpl/absprintf.jsp?arnumber=1212687 (10.01.2007). [Seite 45]

[79] C. Enrique Ortiz. The Generic Connection Framework. Technical report, 2003. http://developers.sun.com/techtopics/mobility/midp/articles/genericframework (25.04.2007). [Seite 92]

[80] Jan Peters, Matthias Pressfreund, Roland Rieke, Taufiq Rochaeli, Björn Steinemann, and Ruben Wolf. PF5: Enforcement of Security Policies within the SicAri-Platform, Version 0.8. Technical report, SicAri Konsortium, 2006. [Seiten 111, 112, 123, 125, 126 und 128]

[81] CC Project. Common Criteria v3.1 Part 1: Introduction and general model. Technical report, CC Project, 2006. http://www.commoncriteriaportal.org/public/files/CCPART1V3.1R1.pdf (21.12.2006). [Seiten 17, 19 und 20]

[82] CC Project. Common Criteria v3.1 Part 2: Security functional requirements. Technical report, CC Project, 2006. http://www.commoncriteriaportal.org/public/files/CCPART2V3.1R1.pdf (21.12.2006). [Seiten 17 und 22]

[83] CC Project. Common Criteria v3.1 Part 3: Security assurance requirements. Technical report, CC Project, 2006. http://www.commoncriteriaportal.org/public/files/CCPART3V3.1R1.pdf (21.12.2006). [Seiten 17, 23 und 24]

[84] CC Project. Common Evaluation Methodology v3.1. Technical report, CC Project, 2006. http://www.commoncriteriaportal.org/public/files/CEMV3.1R1.pdf (21.12.2006). [Seite 17]

[85] Liberty Alliance Project. Liberty Alliance ID-FF 1.2 Specification. Technical report, Liberty Alliance Project, 2006. http://www.projectliberty.org/liberty/content/download/1266/8160/file/liberty-idff-1.2-20050520.zip (09.01.2007). [Seite 46]

[86] Liberty Alliance Project. Liberty Alliance ID-SIS 1.0 Specifications. Technical report, Liberty Alliance Project, 2006. http://www.projectliberty.org/liberty/content/download/2442/15793/file/liberty-id-sis-1.0-20061111.zip (09.01.2007). [Seite 46]

[87] Liberty Alliance Project. Liberty Alliance ID-WSF 2.0 Specifications. Technical report, Liberty Alliance Project, 2006. http://www.projectliberty.org/liberty/content/download/2250/14863/file/liberty-idwsf-2.0-20061004.zip (09.01.2007). [Seite 46]

[88] Liberty Alliance Project. Liberty Alliance Reverse HTTP Binding for SOAP Specification v2.0. Technical report, Liberty Alliance Project, 2006. https://www.projectliberty.org/liberty/content/download/909/6303/file/liberty-paos-v2.0.pdf (11.01.2007). [Seite 48]

[89] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. Technical report, W3C, 1999. http://www.w3.org/TR/html401/ (14.12.2006). [Seite 31]

[90] Nick Ragouzis, John Hughes, Rob Philpott, and Eve Maler. Security Assertion Markup Language (SAML) V2.0 Technical Overview. Technical report, OASIS, 2006. http://www.oasis-open.org/committees/download.php/20645/sstc-saml-tech-overview-2%200-draft-10.pdf (11.01.2007). [Seite 46]

[91] R. Rivest. The MD5 Message-Digest Algorithm. Technical report, Network Working Group Request for Comments, 1992. http://tools.ietf.org/html/rfc1321 (31.12.2006). [Seite 55]

[92] Taufiq Rochaeli and Ruben Wolf. Requirements on SicAri Security Policies, Version 1.0. Technical report, SicAri Konsortium, 2004. http://www.sicari.de/fileadmin/content/Policies/projektarbeiten/pol1_requirements_policies_040525.pdf (06.02.2007). [Seite 100]

[93] R.S. Sandhu. *Role Hierarchies and Constraints for Lattice-Based Access Controls*. Springer-Verlag, 1996. [Seiten 105 und 106]

[94] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32:40–48, 1994. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=7577\&arnumber=312842 (11.01.2007). [Seite 106]

[95] Bruce Schneier. *Angewandte Kryptographie . Protokolle, Algorithmen und Sourcecode in C*. Addison-Wesley, 1996. [Seite 54]

[96] SUN. Connected Limited Device Configuration 1.0 - Specification. Technical report. http://jcp.org/aboutJava/communityprocess/final/jsr030/index.html (02.04.2007). [Seiten 89 und 92]

[97] SUN. Connected Limited Device Configuration 1.1 - Specification. Technical report. http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html (02.04.2007). [Seiten 89 und 92]

[98] SUN. J2ME Building Blocks for Mobile Devices. Technical report, SUN, 2000. http://java.sun.com/products/cldc/wp/KVMwp.pdf (25.04.2007). [Seite 89]

[99] SUN. SATSA Developer's Guide. Technical report, SUN, 2004. http://java.sun.com/j2me/docs/satsa-dg (31.05.2007). [Seite 164]

[100] SUN. CDC: JAVA PLATFORM TECHNOLOGY FOR CONNECTED DEVICES. Technical report, SUN, 2005. http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf (25.04.2007). [Seite 90]

[101] SUN. Connected Device Configuration. Technical report, 2005. http://java.sun.com/j2me/docs/j2me_cdc.pdf (02.04.2007). [Seite 90]

[102] BEA Systems, BMC Software, CA, Inc., IBM, Layer 7 Technologies, Microsoft, Novell, and VeriSign. Web Services Federation Language (WS-Federation) Version 1.1. Technical report, 2006. http://www-128.ibm.com/developerworks/library/specification/ws-fed (23.01.2007). [Seite 83]

[103] Systinet, Microsoft, Sonic Software, BEA Systems, and Computer Associates. Web Services Transfer (WS-Transfer). Technical report, W3C, 2006. http://www.w3.org/TR/ws-addr-core (22.01.2007). [Seite 88]

[104] OASIS XACML TC. OASIS eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS, 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf (23.01.2007). [Seite 86]

[105] Anders Toms. Threats, Challenges and Emerging Standards in Web Services Security. Technical report, University of Skövde, Sweden, 2004. http://episerver.his.se/upload/19352/HS_IKI_TR_04_001.pdf (15.11.2006). [Seite 198]

[106] W3C. XML Path Language (XPath) Version 1.0. Technical report, W3C, 1999. http://www.w3.org/TR/xpath (14.12.2006). [Seiten 31, 58 und 85]

[107] W3C. XSL Transformations (XSLT) Version 1.0. Technical report, W3C, 1999. http://www.w3.org/TR/xslt (14.12.2006). [Seiten 31 und 58]

[108] W3C. Canonical XML Version 1.0. Technical report, W3C, 2001. http://www.w3.org/TR/xml-c14n (02.01.2007). [Seite 58]

[109] W3C. XML Key Management Specification (XKMS). Technical report, W3C, 2001. http://www.w3.org/TR/xkms (23.01.2007). [Seite 86]

[110] W3C. XML Encryption Syntax and Processing. Technical report, W3C, 2002. http://www.w3.org/TR/xmlenc-core/ (02.01.2007). [Seiten 61 und 64]

[111] W3C. XML-Signature Syntax and Processing. Technical report, W3C, 2002. http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ (02.01.2007). [Seiten 58 und 59]

[112] W3C. SOAP Version 1.2. Technical report, W3C, 2003. http://www.w3.org/TR/soap/ (14.12.2006). [Seiten 35 und 37]

[113] W3C. Web Services Architecture Requirements. Technical report, W3C, 2004. http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/ (14.12.2006). [Seite 29]

[114] W3C. XML Schema (Second Edition) W3C Recommendation. Technical report, W3C, 2004. http://www.w3.org/TR/xmlschema-0/ (14.12.2006)http://www.w3.org/TR/xmlschema-1/ (14.12.2006)http://www.w3.org/TR/xmlschema-2/ (14.12.2006). [Seiten 29 und 31]

[115] W3C. Document Object Model (DOM). Technical report, W3C, 2005. http://www.w3.org/DOM/DOMTR (14.12.2006). [Seite 34]

[116] W3C. Extensible Markup Language (XML) 1.1 (Second Edition). Technical report, W3C, 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/ (14.12.2006). [Seite 29]

[117] W3C. Extensible Markup Language (XML) 1.1 (Second Edition) Prolog and Document Type Declaration. Technical report, W3C, 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/sec-prolog-dtd (14.12.2006). [Seite 31]

[118] W3C. Extensible Stylesheet Language (XSL) Version 1.1. Technical report, W3C, 2006. http://www.w3.org/TR/xsl11/fo-section (14.12.2006). [Seite 31]

[119] W3C. Extensible Stylesheet Language (XSL) Version 1.1 - Formatting Objects. Technical report, W3C, 2006. http://www.w3.org/TR/xsl11/ (14.12.2006). [Seite 31]

[120] W3C. Namespaces in XML. Technical report, W3C, 2006. http://www.w3.org/TR/xml-names (21.01.2007). [Seiten 32 und 85]

[121] W3C. Web Services Addressing 1.0 - Core. Technical report, W3C, 2006. http://www.w3.org/TR/ws-addr-core (22.01.2007). [Seite 87]

[122] W3C. Web Services Description Language (WSDL) Version 2.0. Technical report, W3C, 2006. http://www.w3.org/2002/ws/desc/ (14.12.2006). [Seite 37]

[123] Dapeng Wang, Thomas Bayer, Thilo Frotscher, and Marc Teufel. *Java Web Service mit Apache Axis*. Software & Support Verlag 2004, 2004. [Seite 109]

[124] Ruben Wolf and Markus Schneider. Context-dependent Access Control for Web-based Collaboration Environments with Role-based Approach. Technical report, Computer Network Security, 2nd Intern. Workshp on Mathematical Methode, Models and Architectures for Computer Network Security, 2003. [Seite 106]

[125] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol - RFC 4252. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4252 (18.01.2007). [Seite 68]

[126] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol - RFC 4254. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4254 (18.01.2007). [Seite 68]

[127] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture - RFC 4251. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4251 (18.01.2007). [Seite 68]

[128] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocoll - RFC 4253. Technical report, Network Working Group Request for Comments, 2006. http://tools.ietf.org/html/rfc4253 (18.01.2007). [Seite 68]